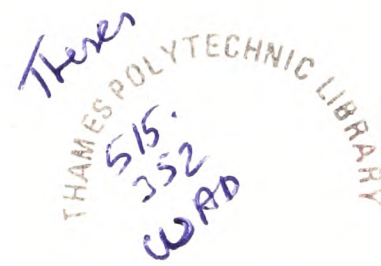
SARK : A TYPE-INSENSITIVE RUNGE-KUTTA CODE

BY

KEVIN CHRISTOPHER WADE
BSc (Hons)

Thesis submitted to the Council for National Academic Awards in partial fulfilment of the requirements for the Degree of Doctor of Philosophy.



Centre for Numerical Modelling and Process Analysis
School of Mathematics, Statistics and Computing
Thames Polytechnic
London

September 1987

TO MY PARENTS

ABSTRACT

A novel solution method based on Mono-implicit Runge-Kutta methods has been fully developed and analysed for the numerical solution of stiff systems of ordinary differential equations (ODE). These Backward Runge-Kutta (BRK) methods have very desirable stability properties which make them efficient for solving a certain class of ODE which are not solved adequately by current methods.

These stability properties arise from applying a numerical method to the standard test problem and analysing the resulting stability function. This technique, however, fails to show the full potential of a method. With this in mind a new graphical technique has been derived that examies the methods performance on the standard test case in much greater detail. This technique allows a detailed investigation of the characteristics required for a numerical integration of highly oscillatory problems.

Numerical ODE solvers are used extensively in engineering applications, where both stiff and non-stiff systems are encountered, hence a single code capable of integrating the two categories, undetected by the user, would be invaluable. The BRK methods, combined with explicit Runge-Kutta (ERK) methods, are incorporated into such a code. The code automatically determines which integrator can currently solve the problem most efficiently. A switch to the most efficient method is then made. Both methods are closely linked to ensure that overheads expended in the switching are minimal. Switching from ERK to BRK is performed by an existing stiffness detection scheme whereas switching from BRK to ERK requires a new numerical method to be devised. The new methods, called extended BRK (EBRK) methods, are based on the BRK methods but are chosen so as to possess stability properties akin to the ERK methods. To make the code more flexible the switching of order is also incorporated.

Numerical results from the type-insensitive code, SARK, indicate that it performs better than the most widely used non-stiff solver and is often more efficient than a specialized stiff solver.

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to my supervisors Dr. Will Richards and Dr. Martin Everett for their interest and support in my work. In particular my appreciation is further extended to Dr. Richards for introducing me to the project, and for his ongoing assistance.

I also wish to thank Professor John Butcher, Dr. Jeff Cash, Dr. Graeme Cooper and Dr. Roland England for their advice and many useful discussions.

I am also grateful to the computer staff at Thames Polytechnic for their continuous help in providing resources, and for their general assistance whenever computer problems arose.

I also wish to thank my family for their support during my studies and all my colleagues and friends at Thames Polytechnic for making my stay there a most enjoyable one. In particular, I am extremely grateful to my fiancée Priti Shah for her careful checking of this manuscript and her constant encouragement.

Finally, I would like to thank the Science and Engineering Research Council for their financial support in all the work reported in this thesis.

NOMENCLATURE

Arg(q)	Argument of complex q
E(q)	General stability function of a Runge-Kutta method
$E_e(q)$	Stability function of ERK method
$E_b(q)$	Stability function of BRK method
eps	Smallest machine representable number such that 1+eps>1
exp(a)	exponentional function, e^a
h	Step size
i	$\sqrt{-1}$
i,j	Superscripts for iteration loops
i,j	Subscripts for loop counts
J	Jacobian matrix of the system under consideration
M	Approximation to the iteration matrix
N	Dimension of ODE system
p	Order of the method
q	step size, h, multiplied by complex λ of scalar test problem
$R_{i,j}$	Padé i,j approximation
Re(q)	Real part of complex q
r	Residual vector
S(x)	Stiffness ratio at position x
s	Number of stages of the method
t,x	Independent variables
x_n	x-value at nth step
y	Dependent variable
y_n	Numerical solution at position x_n
$y(x_n)$	Analytical solution at position x_n
Δ	Displacement vector
ϵ	Error vector

λ	Real or complex scalar
λ_i	General eigenvalues of Jacobian of system
θ	General Angle
μ_i	Eigenvector corresponding to λ_i
$\|a\|$	Modulus of complex a
$\|\|a\|\|$	Norm of a

CONTENTS PAGE

CONTENTS PAGE

CONTENTS **PAGE**

Chapter 1 : INTRODUCTION

In this chapter the general problem to be solved will be defined and sufficient conditions for the existence of a unique solution stated. The general concept of stability and stiffness as applied to Ordinary Differential Equations (ODEs) will be introduced. An outline of some of the methods commonly used in the numerical solution of ODEs will be discussed and finally a brief overview of the remainder of this thesis will be presented.

1.1 The problem considered

This thesis is concerned with the numerical integration of the initial value problem,

$$\begin{aligned} \frac{dy_1}{dx} &= f_1(x, y_1, \ldots y_N) \qquad y_1(a) = \eta_1 \\ &\cdot \qquad\qquad\qquad\qquad\quad \cdot \\ \frac{dy_N}{dx} &= f_N(x, y_1, \ldots y_N) \qquad y_N(a) = \eta_N \end{aligned} \tag{1.1}$$

$$x \geqslant a$$

ie. a system of first order ODEs. Such systems may arise naturally or from reducing a higher order equation to a system of first order equations. Many engineering processes can be expressed mathematically as ODEs, Bjurel et al.[1970], and hence the efficient and accurate numerical solution of such systems plays an important role in industry. By expressing f and y as vectors, (1.1) may be rewritten as

$$\frac{dy}{dx} = f(x,y) \tag{1.2}$$

$$x \geqslant a \ , \ y(a) = \eta$$

Before a numerical solution to (1.2) is obtained it is natural to determine conditions under which a unique solution does exist. For the

initial value problem (1.2) suppose that f(x,y) is continuous in a region D where

$$D = \{ (x,y): a \leqslant x \leqslant b, \|y\| < \infty \} \qquad (1.3)$$

then suppose there exists a finite Lipschitz constant, L, such that

$$\|f(x,y) - f(x,z)\| \leqslant L\|y-z\| \qquad (1.4)$$

for every pair of points (x,y) and (x,z) in D. Then there exists a unique function y(x) which satisfies (1.2), Henrici[1962]. Clearly these conditions are very demanding and can accordingly be weakened to allow a unique solution in some interval |x-a|. Assume that f(x,y) is continuous in some interval D where

$$D = \{ (x,y): |x-a| \leqslant \alpha, \|y-b\| \leqslant \beta \} \qquad (1.5)$$

then suppose there exists a finite constant L such that

$$\|f(x,y) - f(x,z)\| \leqslant L\|y-z\| \qquad (1.6)$$

holds for every pair of points (x,y) and (x,z) in D and let

$$M = \max_{(x,y)\in D} \|f(x,y)\| \qquad (1.7)$$

and

$$\gamma = \min(\alpha, \beta/M) \qquad (1.8)$$

then there exists a unique solution y(x) of (1.2) in the interval $|x-a| \leqslant \gamma$. Repeated use of the above, over a sequence of intervals which together cover the desired integration range, allows a unique solution over the complete range to be proved.

In practice integration of (1.2) is performed by marching from x = a up to some finite b in discrete steps, thus solutions are generated at $a = x_0 < x_1 < \ldots < x_M = b$. Such methods are known as discrete variable methods. A general k-step class of such a method is given by,

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = h\Phi_f(x_n, y_{n+k}, \ldots, y_n, h) \quad n=0(1)M-k \qquad (1.9)$$

given starting values

$$y_i = S_i(h) \quad i=0(1)k-1$$

where $Mh = b-a$, $h = x_{n+1} - x_n$ here assumed constant and the α_js are constant. If Φ_f is independent of y_{n+k} then the method is *explicit* otherwise it is *implicit*. Most of the common discrete variable methods are encompassed in (1.9), eg. selecting

$$k = 1,$$

$$\Phi_f = \sum_{i=1}^{s} c_i f(x_n + b_i h,\ y_n + h\sum_{j=1}^{i-1} a_{ij} k_j)$$

and $\alpha_1 = 1$

(where the constants depend upon the particular method and are defined later) will produce an s-stage, one step (explicit) class of method known as Runge-Kutta methods. The simplest of these being the Euler method,

$$\begin{aligned} y_{n+1} &= y_n + hk_1 \\ k_1 &= f(x_n, y_n) \end{aligned} \qquad (1.10)$$

Generally a solution at y_{n+1} is produced by taking a sample of the function at discrete points between x_n and x_{n+1}, producing a set of s k-values. A linear combination of these k-values are then added to the solution produced at x_n.

The ability of a numerical method to generate a series of solution values at a set of node points is, however, no guarantee that the solution produced is a reasonable approximation to the true solution. The error produced by the method must be investigated. Global truncation error is defined by

$$e_n = y(x_n) - y_n \qquad (1.11)$$

where $y(x_n)$ denotes the true, but usually unknown, solution at x_n. Clearly as the step size of the numerical method is reduced the solution produced by it should approach the true solution ie.

$$\max_{o<n<M-k} \| e_{n+k} \| \to 0 \text{ as } h \to 0 \qquad (1.12)$$

A method which satisfies (1.12) is said to be convergent. From a numerical point of view it is clearly inappropriate to control (1.11) at each step, as the true solution is unknown. The quantity which is usually controlled is the local truncation error, T_{n+k}, of the method. This can be thought of as the error introduced by the formula at each step assuming that no errors have been previously created, ie.

$$T_j = y(x_n) - S_j(h) \quad j=0(1)k-1 \qquad (1.13)$$

$$T_{n+k} = \sum_{i=o}^{k} \alpha_i y(x_{n+i}) - h\Phi_f(x_n, y(x_{n+k}), \ldots, y(x_n), h) \quad n=0(1)M-k$$

The question of 'how accurate is the numerical solution?' can, in part, be answered by considering the order of accuracy of the local truncation error. A method of order p is defined as having

$$\max_{n=0(1)M-k} \| T_{n+k} \| = O(h^{P+1}) \qquad (1.14)$$

and a method of order at least 1 is said to be consistent. Clearly a 'usable' numerical method must be consistent. Consistency, however, does not imply convergence, Hall and Watt[1976].

1.2 Stability

Unfortunately the convergence of a method only deals with the behaviour of the method as h tends to zero and in practice h must be non-zero. Clearly any (stability) constraints of a method will depend upon the problem being solved, thus some standard equation is required. The equation usually considered is the one dimensional test equation,

$$y' = \lambda y \qquad y(0) = 1 \qquad (1.15)$$

where λ may be complex. By applying a Runge-Kutta method to (1.15) a stability function, E(q), is formed which will, in general, be a rational polynominal in $q=h\lambda$. The absolute stability region of a

numerical method is defined to be a region in the complex plane for which E(q) is less than one in modulus. By ensuring that q remains in this stability region then propogated errors will decay as the solution proceeds. If instead of a single equation a linear system of N equations is considered, ie.

$$y' = Jy \qquad y(0) = A \tag{1.16}$$

where J is a constant N×N matrix, A is given and J has eigenvalues λ_i for i=1(1)N, then instead of q we must consider $q_i = h\lambda_i$ for i=1(1)N. We must ensure that for all q_i such that $Re(q_i)$ is less than zero, q_i lies within the region of absolute stability.

When encountering problems for which $Re(\lambda_i)$, for some i, is large and negative, then clearly a finite stability region will restrict the step size of the method. When solving such problems the corresponding q_i must always be included in the stability region of the method. Thus the stability region must include some substantial portion of the left-hand half plane. If the stability region of a method includes the whole of the left-hand half plane then the method is said to be A-stable. A method whose stability region is exactly the left-hand half plane is said to be a precisely A-stable method.

No explicit k-step method can have this property and the highest attainable order of an A-stable implicit Linear Multistep method is two, Dalhquist[1963]. Clearly this stability property places a severe restriction on the numerical method and in particular Linear Multistep methods.

A less severe restriction is that of A(α)-stability where α is an angle in $[0,\pi/2]$, as shown in Figure 1.1

A-stability or A(α)-stability examines the absolute stability region ie. the rate at which the growth of the true solution to (1.15) is modelled by the numerical method. By considering the rate at which the solution grows relative to the exact solution a relative stability region can also be defined. (Unfortunately the term relative stability region is used to mean something quite different in the context of linear multistep methods, Lambert[1973]). Wanner et al.[1978] refer to this region as the order star of the method. The absolute stability region and order star of the unique 1-stage 1st order Runge-Kutta method (Euler) are shown in Figure 1.2.

As Re(q) $\rightarrow -\infty$ in (1.15) the ratio $y(x_{n+1})/y(x_n) \rightarrow 0$, hence a numerical method that is to realistically model this ratio must also produce this behaviour. By applying a Runge-Kutta method to (1.15) and forming the corresponding numerical ratio, y_{n+1}/y_n, the L-stability of that method can be assessed. A method is said to be L-stable, in addition to being A-stable, if when applied to (1.15)

$$\lim_{\mathrm{Re}(q)\rightarrow-\infty} \frac{y_{n+1}}{y_n} \rightarrow 0 \qquad (1.17)$$

holds or L(α)-stable if the method is A(α)-stable.

1.3 Stiffness

In many engineering applications the system of ODEs being integrated possesses both fast and slow transients which must be followed correctly. This phenomenon is known as stiffness and must be correctly modelled by the numerical method. The first formal definition of stiffness was given by Lambert[1973]. A linear system $y' = \Lambda y + \Phi(x)$ is said to be stiff when

i) $Re(\lambda_i) < 0 \quad i=1(1)N$ and

ii) $$S(x) = \max_{i=1(1)N} |Re(\lambda_i)| \; / \; \min_{i=1(1)N} |Re(\lambda_i)| >> 1 \qquad (1.18)$$

where λ_i, $i=1(1)N$ are the eigenvalues of the N×N matrix Λ. This definition can also be used for non-linear systems if the eigenvalues of $\partial f/\partial y$ are considered. The system will then be stiff in an interval I(x) if i) and ii) above are satisfied. The quantity S(x) defines the (local) stiffness ratio of the problem.

This definition is acceptable if it is not taken too literally. It should only be used as a guide, as stiffness is more complicated than this and depends upon the solution method, the problem being solved and the local accuracy requirements. An improved definition of stiffness is that of Shampine[1975], which states that a problem is stiff when the step length is restricted for reasons of stability. But clearly no numerical figures can be attributed to this definition and (1.18) is still useful as the formal definition.

1.4 Numerical integrators

The general class of k-step integration method (1.9) incorporates most of the commonly used methods with the Euler method being the simplest and most basic. This thesis although restricted to Runge-Kutta methods, will use other integrators for comparison purposes, and these are described below along with a review of Runge-Kutta methods.

1.4.1 Linear Multistep methods

A class of methods, based upon past information, are Linear Multistep methods, these have the general form,

$$\sum_{j=o}^{k} \alpha_j y_{n+j} = \sum_{j=o}^{k} \beta_j f_{n+j} \qquad (1.19)$$

If $\beta_k = 0$ ($\alpha_k \neq 0$) then the method is explicit otherwise it is implicit. When such a method is explicit then (1.19) can be solved directly otherwise some iterative scheme must be employed. One class of linear multistep methods commonly used for the numerical integration of non-stiff problems are Adams methods. These methods are derived by replacing the function in (1.2) by a polynomial and integrating this, Shampine and Gordon[1975].

The Adams methods incorporated in the NAG library have explicit predictors, chosen to maximize the stability region, and implicit Adams-Moulton methods for the corrector in a PECE implementation. The implicit method is solved by means of a simple functional iteration and the error estimation is performed by Milne's device. The NAG implementation incorporates methods of orders one to twelve.

The most commonly used methods for solving stiff systems are the Backward Differentiation Formulae (BDF) popularized by Gear[1971]. These methods have the general form

$$\sum_{j=o}^{k} \alpha_j y_{n+j} = h\beta_k f_{n+k} \qquad (1.20)$$

Although these are k-step methods, they only require one function evaluation per iteration at each step. The implicit equations are solved by a Quasi-Newton method. The BDF methods, orders one to five, are used in conjunction with starting values obtained by extrapolation using a divided difference table. The major handicap with BDF methods is that their stability properties deteriorate as the order is increased. When applied to (1.15), BDF methods of order greater than six are not A(0)-stable and hence they are of little value. Although the number of function evaluations required is low their overheads are

high. Craigie[1975] describes in detail the complexity of a modern version of Gear's method.

1.4.2 Runge-Kutta methods

The general form of an s-stage Runge-Kutta method is

$$y_{n+1} = y_n + h\sum_{i=1}^{s} c_i k_i \qquad (1.21)$$

$$k_i = f(x_n + hb_i,\ y_n + h\sum_{j=1}^{s} a_{ij}k_j) \quad i=1(1)s$$

The constants a_{ij} and c_i characterise the particular method and

$$b_i = \sum_{j=1}^{s} a_{ij} \qquad (1.22)$$

The coefficients can be expressed in terms of a matrix system, called the Butcher matrix of the method. This is

$$\begin{array}{c|ccc} b_1 & a_{11} & \cdots & a_{1s} \\ \vdots & \vdots & & \vdots \\ b_s & a_{1s} & \cdots & a_{ss} \\ \hline & c_1 & \cdots & c_s \end{array} \quad \text{or} \quad \begin{array}{c|c} b & A \\ \hline & c^T \end{array} \qquad (1.23)$$

where the s×s matrix A is strictly lower triangular for an explicit method, lower triangular for a semi-implicit (or semi-explicit) method and full for a fully implicit method.

Due to their simplicity, explicit methods have been very popular and high order methods have been derived. The minimum number of stages required to solve the resulting non-linear order constraint equations is shown in Table 1.1. A * by the number of stages denotes that the minimum number of stages is unproven but methods with this number of stages have been derived.

As will be shown in section 2.2, no explicit Runge-Kutta method can

possess an infinite stability region and hence the step must be severely restricted when solving problems with fast transients. For this reason implicit Runge-Kutta (IRK) methods have become very attractive, as they can be A-stable for high orders. Ehle[1968] proved that an s-stage 2s order IRK method can be A-stable. However IRK methods suffer from a severe practical disadvantage. If an s-stage method is used to solve (1.2), then a system of sN implicit algebraic equations have to be solved at each step. By using the Newton iteration process this involves approximately s^3N^3 multiplications for the LU factorization of the iteration matrix and s^2N^2 multiplications for the back solvers. This is clearly expensive, expecially for high order methods.

An enormous improvement in computational efficiency can be achieved if semi-implicit methods are used, Alt[1972], Norsett[1974], Crouziex[1976] and Alexander[1977]. By using semi-implicit methods the process at each step involves the solution of s systems of N algebraic equations. In solving the algebraic equations an iteration matrix of the form

$$I-ha_{ii}\partial f/\partial y \qquad (1.24)$$

must be evaluated, where the a_{ii}'s are the diagonal elements of the Butcher matrix. In a semi-implicit method $\partial f/\partial y$ will be calculated, and stored, and (1.24) evaluated for each different a_{ii}. But by selecting all the a_{ii} values the same (1.24) need be evaluated only once, ie. the method has only one s-fold zero of the stability function. Such methods are known as Diagonally Implicit Runge-Kutta methods (DIRK), Alexander[1977]. However, a semi-implicit method can have at most order s+1.

Cash[1975] derived a type of Runge-Kutta method that is a significant departure from traditional methods. These methods are implicit in the single unknown y_{n+1} and not in the k values like IRK methods. The general form of the s-stage method is,

$$y_{n+1} = y_n + h\sum_{i=1}^{s} c_i k_i$$

$$k_i = f(x_n + hb_i,\ y_n + h\sum_{j=1}^{r} a_{ij}k_j) \qquad i=1(1)r \qquad (1.25)$$

$$k_i = f(x_{n+1} + hb_i,\ y_{n+1} + h\sum_{j=r+1}^{s} a_{ij}k_j) \qquad i=r+1(1)s$$

By being implicit in only y_{n+1} only one set of algebraic equations needs to be solved at each step. These Mono-Implicit Runge-Kutta (MIRK) methods, require only one LU factorization and s back substitutions, Singhal[1980]. Two important class of methods are included in MIRK methods, viz. explict Runge-Kutta, r=s, and Backward Runge-Kutta, r=0. These Backward Runge-Kutta (BRK) methods will be analysed in detail in this thesis.

1.5 Selection of an appropriate numerical method

When the numerical solution of (1.2) is required the user has a vast bank of methods to select from. These range from low order to high order, explicit or implicit methods of either single-step or multistep or of one of the more unusual methods ie. Rosenbrock, Block implicit Runge-Kutta, etc. The method chosen must be capable of integrating the problem efficiently ie. accurately and within a reasonable CPU time.

The problem of selecting an integrator for the whole integration range is two-fold, firstly if the incorrect method is used the integration will be inefficient. Secondly the characteristics of the problem may, and often do, change during the integration range.

Clearly no single numerical scheme (where scheme implies the complete solution algorithm, ie. numerical integrator and if relevant the linear equation solver) can possess the correct characteristics to enable it to efficiently solve non-stiff and stiff ODEs.

A simple solution, is to always employ an implicit method with the implicit equations being solved by a Newton type process. This will, however, be inefficient for the non-stiff problems.

A better solution is to use a numerical scheme (integrator plus linear equation solver) that monitors the characteristics of the problem and can automatically detect changes in these characteristics and switch to a scheme that is most appropriate for the problem at that particular time. Codes that can automatically do this are often referred to as type-insensitive.

There are two basic switching strategies;

i) incorporate two integrators in a code and switch between the two or

ii) employ only one basic implicit integrator and switch the iteration process for solving the implicit equations.

Both methods have been investigated and production codes developed. Petzold[1983] produced a code that switched between Adams and BDF methods. As stated earlier the main drawback with BDF methods is their order limitation for practical purposes, they can not be greater than 5. The overheads in linear multistep methods are high and so are the overheads in switching.

The code of Norsett and Thomsen[1986] keeps the same numerical

integrator, an implicit Runge-Kutta method, and switches the implicit equation solver. For the non-stiff case simple functional iteration is used whereas Quasi-Newton is employed for the stiff case. This has the disadvantage that some iterative scheme must always be employed, which is expensive. The code is also restricted to a fixed order.

1.6 Overview of the thesis

This thesis is concerned with the development of numerical schemes for the solution of initial value ODEs. A new graphical technique for assessing the performance of potential methods is described in chapter 2, with particular attention to highly oscillatory problems.

Chapter 3 develops the theory behind Backward Runge-Kutta methods and in particular their close coupling with explicit Runge-Kutta methods. It also shows that they have far superior damping properties than the most widely used stiff solvers. Numerical examples are presented, without the hinderance of error control, that shows the potential of the methods.

In chapter 4 the error control policies applicable to BRK methods are explored and it is shown why the normal embedding method, commonly used for explicit methods, cannot be employed in the BRK case. The error control policy adopted is discussed and incorporated into the code and compared with the BDF code implementation of the NAG library.

Most of the numerical integrators incorporated in codes suffer from some inefficiencies when solving a certain type of problem. It is well known that BDF methods are extremely inefficient for solving problems which possess highly oscillatory solutions. Chapter 5 discusses the

class of problem for which BRK methods are inefficient.

Chapter 6 develops the strategies for switching between explicit and Backward Runge-Kutta methods. Thus a type-insensitive Switching Algorithm for Runge-Kutta methods (SARK) is devised. The switching of order is also discussed and implemented in the final code. Numerical examples are given that highlight the necessity for a code of this type.

When developing any numerical code for the solution of ODEs it is impossible to test the code on all systems of ODEs and hence a test battery is required. The test battery that is commonly used is the DETEST set of Enright and Pryce[1983]. The code developed in chapter 6, SARK, is compared with the BDF code over the stiff and non-stiff problems of the set. As BDF methods are not designed to integrate non-stiff systems the Adams methods, used in the NAG library, are also tested and compared with SARK over the non-stiff set.

Order	1	2	3	4	5	6	7	8	9	10	11
Stages	1	2	3	4	6	7	9	11	16*	17*	----
Equations to solve	1	2	4	8	17	37	85	200	486	1205	3047

Table 1.1 : Minimum number of stages for each order.

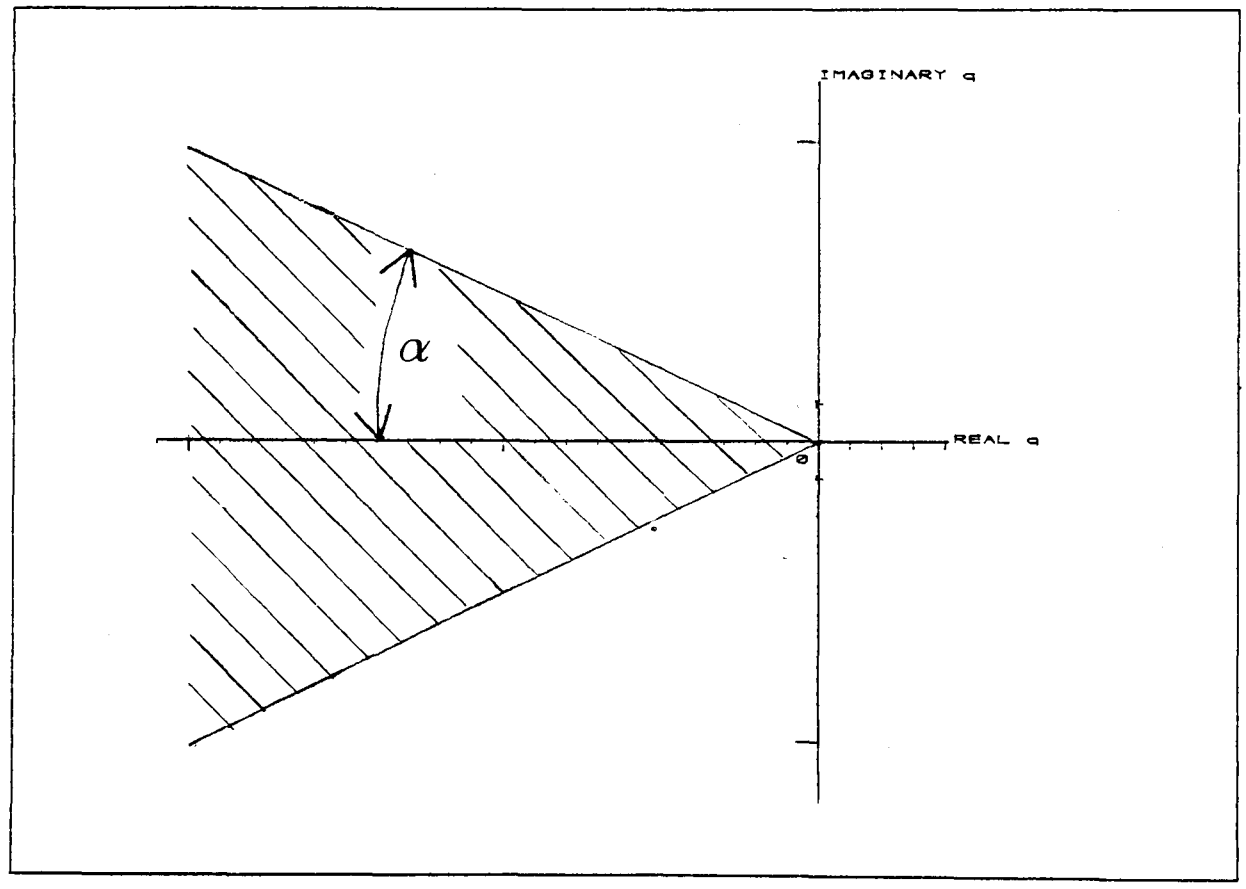


Figure 1.1 : A(α)-stability region.

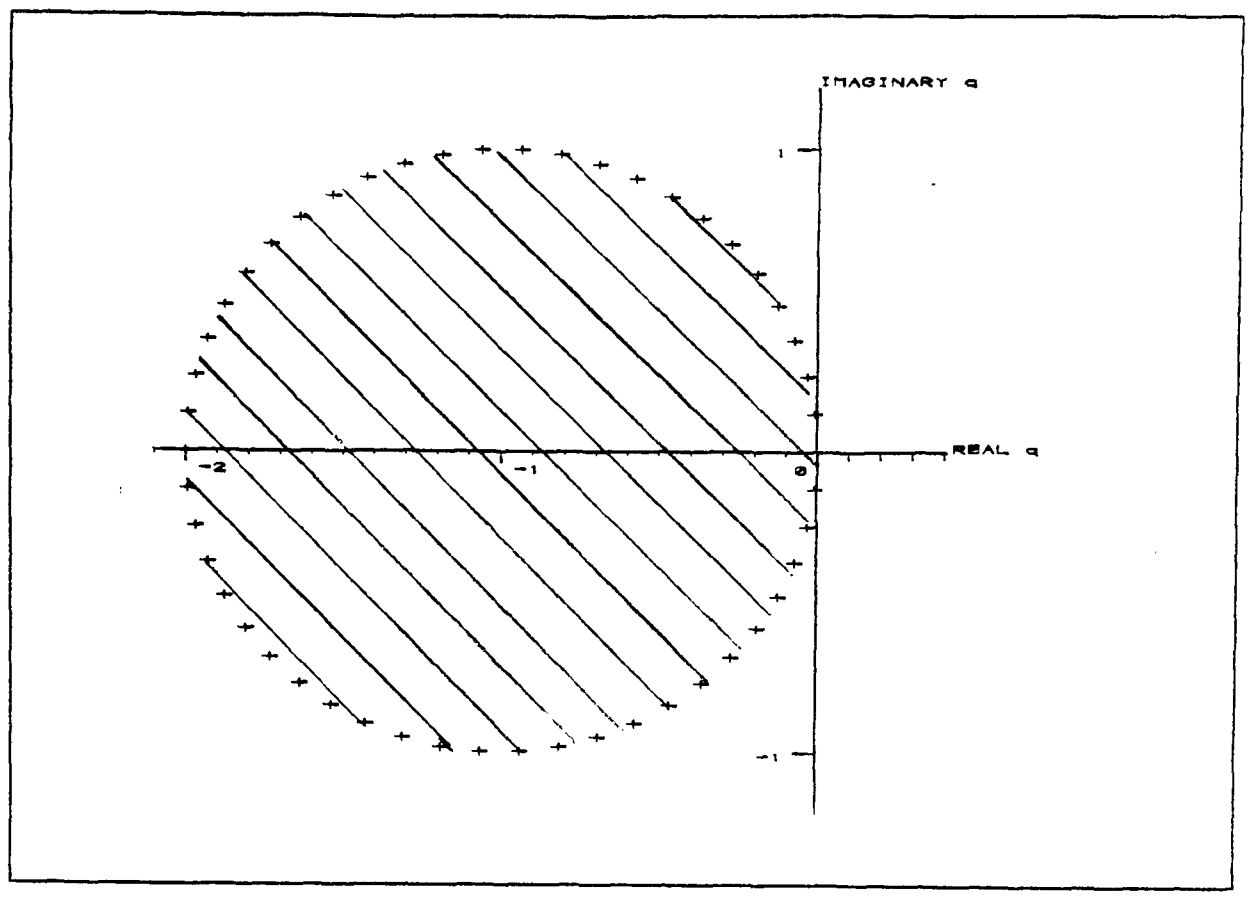


Figure 1.2a : Absolute stability region of Euler's method.

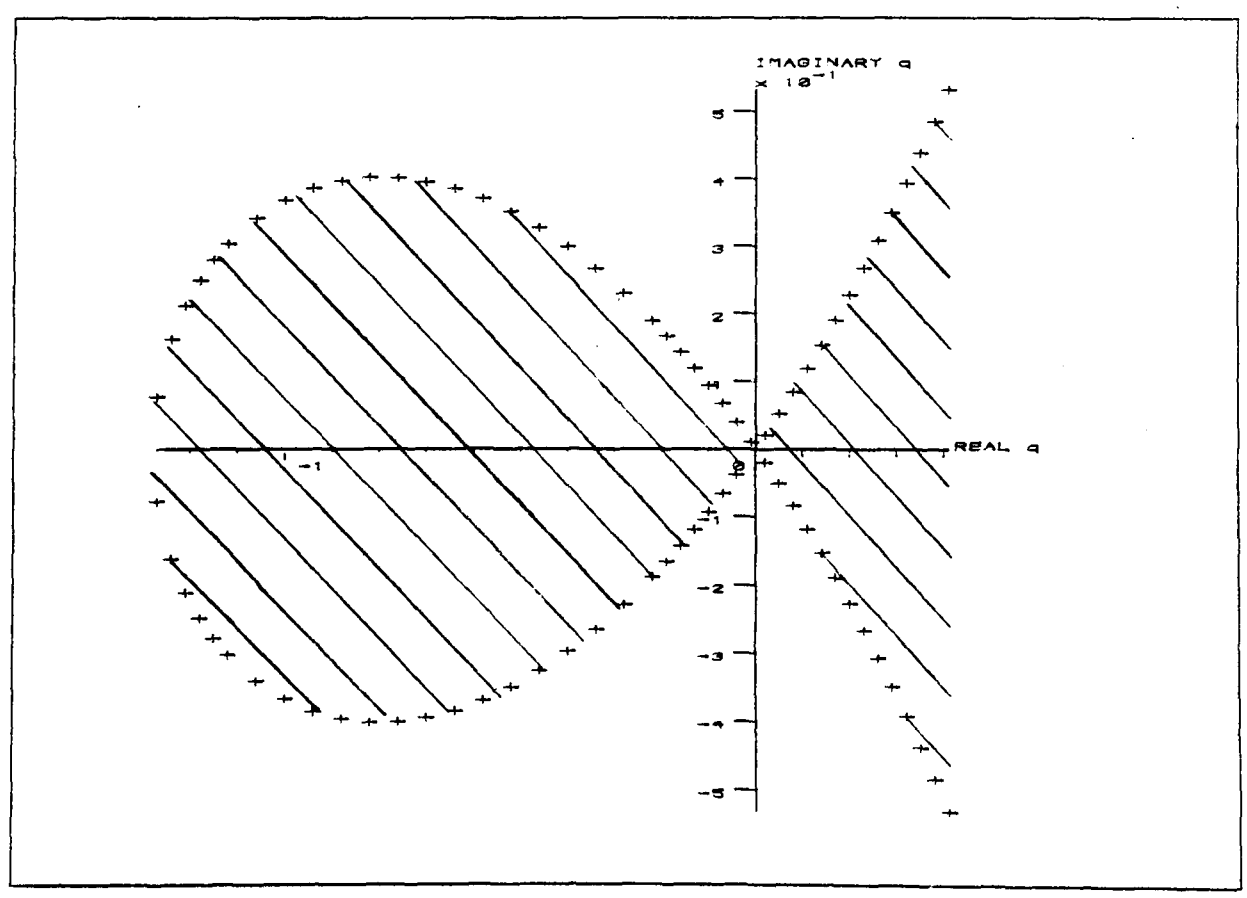


Figure 1.2b : Relative stability region of Euler's method.

Chapter 2 : PREDICTING PERFORMANCE

This chapter addresses the problem of assessing the potential performance of a numerical method, over a wide range of problems. To fully assess the performance of any method for solving initial value problems, it must be fully implemented and applied to a large collection of test problems. To compare a number of methods in this way is clearly a lengthy process. Furthermore, minor changes in the implementation strategy can lead to dramatic improvements or to severe deterioration, making comparisons difficult to interpret. Consequently a quick to use assessment of potential performance, which is independent of algorithmic details, is extremely valuable. This can be used as a sieve to make an initial selection of promising methods which can then be implemented and fully tested on a batch of test problems. A new graphical technique is devised that allows this by comparing the numerical approximation with the exponential solution of the standard test problem in much greater detail than existing techniques. This method is extremely quick and easy to perform.

If the ODE being integrated is characterised by imaginary eigenvalues, often giving rise to a highly oscillatory component, then the absence of A-stability in a numerical method has prompted many authors to dismiss it as being inadequate. This new technique introduced gives more insight into this case and as a result this assumption is shown to be invalid.

2.1 Extension of regions of absolute stability

The simple idea of a region of absolute stability has been extensively used for assessing methods. The stability region gives some insight into the stability characteristics of a numerical method when solving

systems of ODEs. Integrating with q (=hλ) within the stable region is, however, no guarantee that the solution produced will model realistically the solution of the system. Indeed if Re(q) is greater than zero, it could be disasterous to integrate with q within this region.

Recall the standard test problem,

$$\frac{dy}{dx} = \lambda y \qquad y(0)=1 \qquad (2.1)$$

which has the analytical solution

$$y(x) = \exp(\lambda x) \qquad (2.2)$$

If the analytical solution is examined at a series of node points x_n = nh for n = 0, 1, . . . then

$$\frac{y(x_{n+1})}{y(x_n)} = \exp(q) \qquad (2.3)$$

When the numerical method is applied to (2.1) with constant step h, the corresponding numerical ratio is

$$\frac{y_{n+1}}{y_n} = E(q) \qquad (2.4)$$

This ratio is the stability function of the method and is a numerical approximation to (2.3). The region of absolute stability of the method is defined as being the region(s) of the complex plane where propagated errors decay as the solution proceeds. One way to identify the stability region of a method is to find its boundary. It can easily be verified that the boundary is generated from the stability function by equating its modulus to unity, ie. |E(q)|=1. One such technique for locating this boundary is the boundary locus method Lambert[1973].

Generally E(q) ≠ exp(q), but it is hoped that E(q) ≈ exp(q). The absolute stability region gives only limited indication as to what

extent the numerical ratio is a good approximation to the analytical one.

By expressing

$$q = a + ib \qquad (2.5)$$

in (2.3), the analytical ratio can be written as

$$\frac{y(x_{n+1})}{y(x_n)} = \exp(a+ib) = \exp(a)\times\{\cos(b) + i\sin(b)\} \qquad (2.6)$$

where e^a is a measure of the damping of the component and b, the argument of q, is its frequency. If a is less than zero the solution will decay to zero whereas if a is greater than zero the solution grows in amplitude. The stability function, E(q) should approximate both the damping and the frequency of the component to produce realistic results. It follows that we require the approximate relation between the complex quantities ie.

$$E(q) \approx \exp(a+ib) \qquad (2.7)$$

to be good in terms of both modulus and argument. This will ensure that both damping and frequency are realistic. Therefore it is necessary to consider two aspects of the approximation (2.7), viz. the damping and the frequency.

Analysis of the damping characteristics of a method can be performed by comparing the modulus of the stability function with the modulus of the analytical ratio (2.6). Hence we require,

$$|E(q)| \approx |\exp(a+ib)| = \exp(a) \qquad (2.8)$$

By expressing E(q) as $R\exp(i\theta)$

$$|E(q)| = R \qquad (2.9)$$

Therefore from (2.8) and (2.9)

$$R \approx \exp(a) \qquad (2.10)$$

is required. Thus numerical contours expressing the damping characteristics of the method can be produced by plotting q such that

$$E(q) = R \qquad (2.11)$$

for various values of R. These can then be compared with the analytical contours for which exp(a) = R. The latter, from (2.8) are straight lines logarithmically spaced perpendicular to the real axis.

The ability of a numerical method to model realistically the frequency of a component can be determined by comparing arg(E(q)) with arg(exp(q)). Using (2.5) and expressing E(q) as $R\exp(i\theta)$ then,

$$\arg(\exp(q)) = \arg(\exp(a+ib)) = b \qquad (2.12)$$

and

$$\arg(E(q)) = \arg(R\exp(i\theta)) = \theta \qquad (2.13)$$

Therefore the frequency of the numerical solution is θ which should be a satisfactory approximation to b. Hence numerical contours can be produced and compared with the analytical solution in which the contours are linearly spaced perpendicular to the imaginary axis.

For all Runge-Kutta methods E(q) is a rational polynominal of the form,

$$E(q) = N(q) \,/\, D(q) \qquad (2.14)$$

where N(q) and D(q) are polynomials in q and $D(q) \equiv 1$ for an explicit method. Substituting $R\exp(i\theta)$ for E(q) in (2.14) yields an expression of the form

$$[N(q) - R\exp(i\theta)D(q)] = 0 \qquad (2.15)$$

This polynominal equation with complex coefficients can now be solved for q to produce the contours. By taking a series of R values eg. R = .25, .5, 1., 2., 3. and for each value of R varying θ in the range $0 < \theta < 2\pi$ eg. $\theta = 2\pi j/100$ for j =1(1)100 a series of contours of equal R ie. equal |E(q)| can be generated. Simliarly if θ is fixed at a number

of convenient levels eg. $\theta = -3\pi/4, -\pi/2, -\pi/4, 0, \pi/4, \pi/2, 3\pi/4$ and for each fixed θ solving (2.15) for (complex) q with R = 0.1j for j = 1(1)100, contours of equal arg(E(q)) can be plotted. In each case a polynomial in q must be solved which has complex coefficients. The NAG subroutine C02ADF can be used for this. This technique can be thought of as a logical extension of the boundary locus method.

To illustrate this contouring technique a collection of 4th order Runge-Kutta methods, whose stability functions are Padé approximations are examined. The five approximations considered are:

$$R_{4,0} = 1 + q + q^2/2 + q^3/6 + q^4/24$$

$$R_{3,1} = (1 + 3q/4 + q^2/4 + q^3/24)/(1 - q/4)$$

$$R_{2,2} = (1 + q/2 + q^2/12)/(1 - q/2 + q^2/12) \qquad (2.16)$$

$$R_{1,3} = (1 + q/4)/(1 - 3q/4 + q^2/4 - q^3/24)$$

$$R_{0,4} = 1/(1 - q + q^2/2 - q^3/6 + q^4/24)$$

These approximations, with the exception of $R_{2,2}$, stem from infinite families of methods typified by; the classical 4-stage 4th order explicit method ($R_{4,0}$), Lobatto IIIc method ($R_{3,1}$), Chipmann[1971] and a 4-stage 4th order backward method ($R_{0,4}$). The $R_{2,2}$ approximation is defined uniquely from the 2-stage 4th order fully implicit method which has Butcher matrix shown in Table 2.1.

The only 4th order Padé approximation not in common use as a Runge-Kutta method is the $R_{3,1}$. This approximation can only be derived from a fully implicit method and it possesses a finite stability region and is hence of no practical value.

The modulus and argument plots for these five Padé approximations are shown in Figures 2.1 to 2.5. As all the plots are symmetric about the

real axis, section 3.3, only the positive imaginary axis is displayed. The contours for the modulus plots are presented at five different levels of R, viz. R = 1/4, 1/2, 1.0, 2.0 and 3.0, each contour is represented by a different symbol on the diagram. The argument plots are shown for $\theta \in [\pi/2, 2\pi]$ in intervals of $\pi/2$, again each contour level is denoted by a different symbol. Both sets of analytical contours are superimposed on to the corresponding plot and their value denoted by the symbol located at one end of the contour. The normal region of absolute stability can be observed by considering the contour R = 1 of the modulus plot.

One other desirable stability property required by a numerical method when solving stiff systems is L-stability (chapter 1). The modulus plot has the added advantage of determining whether this property is present in the method. To be L-stable the contours of Re(q) at $-\infty$ must be zero, hence the value of the contours should decrease as Re(q) tends to $-\infty$.

Modulus and argument plots for the Padé $R_{4,0}$ approximation are shown in Figure 2.1. The modulus plot clearly indicates that the method is more successful at producing the correct damping (amplification) for Re(q) greater than zero than for Re(q) less than zero. This is due to the zeros of the stability function being in the left-hand half plane with one close to each of the axes. As q approaches any of the zeros the approximation becomes highly inaccurate. From the argument plot it is clear that the zero close to the imaginary axis will distort the frequency in this region. Also computing with q at 4i will result in the solution being underdamped, whereas with q at 2.5i, within the absolute stability region, results in an overdamped solution.

Therefore the absurdity of the common assumption that computing with q within the absolute stability region guarantees a realistic solution is immediately clear from these plots.

The two plots generated by the $R_{3,1}$ implicit method are shown in Figure 2.2. As this is a rational approximation there are now three zeros and a pole, the pole being on the positive real axis. Again this approximation is more successful at producing the correct damping for Re(q) greater than zero than for Re(q) less than zero, providing that Re(q) is kept away from the pole. The pole and zeros again produce distortions in the two sets of contours, however as they are further away from the imaginary axis the method is more successful for problems with eigenvalues close to this axis. The argument plot highlights the inability of the method to correctly represent the frequency as q departs from the origin.

By considering only the modulus plot of the $R_{2,2}$ approximation, Figure 2.3, it appears that the method is almost ideal for problems with purely imaginary eigenvalues. The analytical contour is followed exactly on this axis. In other words the corresponding method is precisely A-stable, however, the contours in the negative half-plane indicate that it is not L-stable. The argument plot reveals that even though the poles and zeros are well away from the imaginary axis, the frequency will only be modelled realistically for small q. This demonstrates that precise A-stability is not a particularly valuable attribute for solving oscillatory problems.

The next two approximations, $R_{1,3}$ and $R_{0,4}$ are mirror images about the imaginary axis of $R_{3,1}$ and $R_{4,0}$ respectively with the zeros replaced by

the poles and vice versa. These are shown in Figures 2.4 and 2.5 respectively. From the modulus plot it is apparent that the $R_{1,3}$ approximation is A-stable and that they are both L-stable. Both approximations are more successful at producing the correct damping for the Re(q) less than zero than for Re(q) greater than zero, providing that the zero of $R_{1,3}$ at q = - 4 is avoided. The argument plots show that being able to produce the correct damping for Re(q) less than zero is not sufficient to produce realistic results. The step size of both must be restricted to faithfully follow the frequency of the component.

2.2 Application to highly oscillatory problems

The ability of this contouring technique to predict the performance of numerical methods can be demonstrated by considering a class of problem in which the dominant eigenvalues of the Jacobian matrix, $\partial f/\partial y$, are of the general form $a \pm ib$, where $|b/a|$ is much greater than one. Such problems frequently arise in engineering situations and will severely tax any numerical method. This type of problem is often described as highly oscillatory due to dominant eigenvalues of linear problems giving rise to a solution of the form

$$\exp(ax)\sin(bx + c) \qquad (2.17)$$

c constant. This leads to the component having a frequency of $b/2\pi$ Hz. Irrespective of whether the problem is linear, the stability characteristics of the integrator are clearly of importance. It has long been understood, Prothero and Robinson[1974], Jeltsh[1978], Singhal[1980], Gear[1981], that A-stable methods must be employed for such problems.

If only error propagation is considered, then A-stability appears desirable if not essential. But the ability to produce the correct

damping and frequency is also of great importance. It is of no value producing stable results that are physically unrealistic.

The modulus and argument plots clearly show that precisely A-stable methods will need to restrict the step size to follow any high frequency component, as indeed will all the methods. None of the 4th order methods examined will allow a significantly larger step to be used than another. Therefore the method that is "cheapest" computationally must be employed, which is the explicit method. Lack of A-stability will not hinder the method when solving problems with imaginary eigenvalues.

These predictions can be analysed further by considering a variety of Runge-Kutta methods applied to the highly oscillatory problems. Three types of Runge-Kutta method, derived from the same coefficients, are considered. These are outlined below:

(i) Explicit Runge-Kutta (ERK)

The general form of an s-stage ERK method is

$$y_{n+1} = y_n + h\sum_{j=1}^{s} c_j k_j$$

$$k_j = f(x_n + hb_j,\ y_n + h\sum_{i=1}^{j-1} a_{ji} k_i)\quad j=1(1)s \qquad (2.18)$$

and their stability functions are of the form

$$E_e(q) = 1 + \sum_{j=1}^{s} \delta_j q^j \qquad (2.19)$$

where the value of δ_j, $j = 1(1)s$ depends upon the chosen method and in particuilar, $\delta_j = 1/j!$ for any s-stage s order method, ie. s is less than five. Clearly

$$\lim_{Re(q) \to -\infty} |E_e(q)| = \infty \qquad (2.20)$$

and hence no ERK method can be A-stable.

(ii) Backward Runge-Kutta (BRK)

The general form of an s-stage BRK method is,

$$y_{n+1} = y_n + h\sum_{j=1}^{s} c_j k_j$$

$$k_j = f(x_{n+1} - hb_j,\ y_{n+1} - h\sum_{i=1}^{j-1} a_{ji} k_i)\quad j=1(1)s \qquad (2.21)$$

Thus BRK methods can be considered as ERK methods integrating from x_{n+1} to x_n with a step of -h, ie. Backward. Therefore any coefficients from a ERK method can be used to form the corresponding Backward method. Their stability functions, as derived in section 3.1, are of the form,

$$E_b(q) = \frac{1}{1 + \sum_{j=1}^{s} \delta_j (-q)^j} = \frac{1}{E_e(-q)} \qquad (2.22)$$

where the value of δ_j, $j = 1(1)s$ are those of the corresponding ERK method. A-Stable BRK methods of order up to two, are known, with higher order methods being A(α)-stable with α close to 90°. Typical α values attainable are given in Table 2.2, along with the corresponding α values for the well known BDF methods.

(iii) Mixed Runge-Kutta (MRK)

These are derived by alternately using ERK and BRK methods. First the ERK method is applied with step h/2 followed by the corresponding BRK method with the same step. The order of the resulting method is usually the same as the main ERK method but can be higher, (the explicit method which generates the mixed method will be referred to as the main method). For example coupling 1st order Euler with its corresponding BRK method, Backward Euler, gives rise to the precisely A-stable 2nd order Trapezoidal rule.

The stability function of a Runge-Kutta method is generated by applying

the method to the standard test problem, (2.1), with constant step h. Thus for MRK method, this is

$$y_{n+\frac{1}{2}} = E_e(q/2)y_n \tag{2.23}$$

for the first half step using the ERK method and for the second half step using the corresponding BRK method,

$$y_{n+1} = E_b(q/2)y_{n+\frac{1}{2}} \tag{2.24}$$

Hence merging (2.23) and (2.24) and using the result of (2.22)

$$\frac{y_{n+1}}{y_n} = Ee(q/2)E_b(q/2)$$

$$= \frac{E_e(q/2)}{E_e(-q/2)} \tag{2.25}$$

Thus the stability function of a MRK method has the form

$$E_m(q) = \frac{1 + \sum_{j=1}^{s}\delta_j(q/2)^j}{1 + \sum_{j=1}^{s}\delta_j(-q/2)^j} \tag{2.26}$$

For a MRK method the imaginary axis always forms part of the boundary of the region of absolute stability. This can be shown by considering $q = ib$ in (2.26). Hence

$$E_m(ib) = \frac{1 + \sum_{j=1}^{s}\delta_j(ib/2)^j}{1 + \sum_{j=1}^{s}\delta_j(-ib/2)^j}$$

$$= \frac{1 + \sum_{j=2}^{s_1}\delta_{2j}(-1)^j(b/2)^j + i\sum_{j=1}^{s_2}\delta_{2j+1}(-1)^{2j+1}(b/2)^{2j+1}}{1 + \sum_{j=2}^{s_1}\delta_{2j}(-1)^j(-b/2)^j + i\sum_{j=1}^{s_2}\delta_{2j+1}(-1)^{2j+1}(-b/2)^{2j+1}} \tag{2.27}$$

where

$$s_1 = \begin{cases} s/2 \text{ for } s \text{ even} \\ (s-1)/2 \text{ for } s \text{ odd} \end{cases}$$

$$s_2 = \begin{cases} (s-2)/2 \text{ for } s \text{ even} \\ (s-1)/2 \text{ for } s \text{ odd} \end{cases}$$

Clearly as the numerator and denominator are a conjugate pair their moduli are the same, ie.

$$|E_m(ib)| = \frac{|E_e(ib/2)|}{|E_e(-ib/2)|} = 1 \qquad (2.28)$$

and hence the MRK method is stable along the entire imaginary axis. This does not mean, however, that all MRK methods are precisely A-stable, the following theorem demonstrates this.

Theorem 2.1 : A MRK method is precisely A-stable $\Leftrightarrow$ the zeros of the stability function of the main ERK method are all in the left-hand half plane.

Proof : Let the zeros of the stability function of the ERK method be q_j, j=1(1)s. Where $Re(q_j)$ is less than zero for all j, ie. all zeros are in the left-hand half plane. Then, as q_j for all j are roots of (2.19)

$$E_e(q) = \frac{(q_1-q)(q_2-q)\ldots(q_s-q)}{q_1q_2\ldots q_s} \qquad (2.29)$$

From (2.25), the stability function of the resulting mixed method is

$$E_m(q) = \frac{E_e(q/2)}{E_e(-q/2)}$$

$$= \frac{(2q_1-q)(2q_2-q)\ldots(2q_s-q)}{(2q_1+q)(2q_2+q)\ldots(2q_s+q)} \qquad (2.30)$$

which has zeros at $q = 2q_j$ for j=1(1)s which are also in the left-hand half plane. There are poles at $q = -2q_j$ for j=1(1)s which are all in

the right-hand half plane. If follows from the maximum modulus theorem that maximum of $|E_m(q)|$ in the left-hand half plane occurs on the boundary of the region ie. on the imaginary axis. However, from (2.28), the stability function equals unity on this axis and hence,

$$|E_m(q)| < 1 \qquad (2.31)$$

for Re(q) less than zero, ie. it is precisely A-stable if all the zeros of the stability function of the main ERK method are in the left-hand half plane.

Now assume that there exists a precisely A-stable MRK for which the main ERK has a zero in the right-hand half plane, at q_p. This implies, from (2.22), that the corresponding BRK method has a pole at $-q_p$, ie. in the left-hand half plane. hence

$$\lim_{q \to q_p} |E_e(q)| = 0 \text{ and } \lim_{q \to -q_p} 1/|E_e(-q)| = \infty \qquad (2.32)$$

thus

$$\lim_{q \to -2q_p} E_m(q) = \lim_{q \to -2q_p} \frac{E_e(q/2)}{E_e(-q/2)} = \infty \qquad (2.33)$$

Thus a mixed method can only be precisely A-stable if all the zeros of the main ERK method satisfy $Re(q_j)$ less than zero for all j=1(1)s.

All s-stage s order ERK methods give rise to precisely A-stable MRK methods. The location of the zeros for these ERK methods are given in Table 2.3. To highlight the fact that not all ERK methods produce precisely A-stable MRK methods the 6-stage 5th order method of Fehlberg, in MRK mode is shown in Figure 2.6, with the stability region shaded. The zeros of the main ERK method are given in Table 2.4 and are clearly not all in the left-hand half plane resulting in a MRK method which is not precisely A-stable.

A set of Runge-Kutta methods can thus be implemented in three separate modes viz. ERK, BRK and MRK with each possessing very different characteristics.

The highly oscillatory case is under consideration. This is characterised by dominant eigenvalues which are essentially imaginary, hence the modulus and argument plots will be restricted to the imaginary axis. On this axis $|\exp(q)| = 1$ and $\arg(\exp(q))$ has a saw-tooth profile of period 2π.

Figure 2.7 shows these new modulus and argument plots for Euler (ERK), Backward Euler (BRK) and the Trapezoidal rule (MRK). The latter being precisely A-stable produces the correct damping for all imaginary q. The ERK, on the other hand, underdamps the component and the BRK overdamps as q departs from the origin. The $\arg(\exp(q))$ appears on the argument plot as the saw-tooth profile. Clearly the ability of the MRK to reproduce the correct damping is offset by its inability to correctly predict the frequency for large imaginary q. Clearly $\arg(E(q))$ is the same for both ERK and BRK modes, and as shown their ability to accurately represent the frequency is limited to small q values.

Modulus and argument plots for methods based on 3rd order ERK methods are shown in Figure 2.8. It is readily apparent that as the order increases the range of q for which the correct damping and frequency can be reproduced also increases. The argument plot clearly shows that the MRK method can correctly follow the frequency for twice the value of that allowed by the ERK or BRK methods. This indicates that the MRK method can be used with twice the step size of that allowed for by the

other modes. It must be remembered, however, that the MRK method requires, at best, twice as many function evaluations per step as the corresponding ERK method, (and may be more if repeated iterations are required to solve the implicit equations).

The ability to extend the range of q for which the correct characteristics are produced when the order is increased, can be illustrated by considering the 5th order 6-stage method of Fehlberg and the 8th order 12-stage method of Verner[1978], Figures 2.9 and 2.10 respectively. As q increases along the imaginary axis the ERK methods eventually underdamp the solution, modulus plots. Before they do this however, they overdamp for a small range of q. This is due to zeros of the stability function distorting the contours in these regions. Similarly the Backward methods underdamp in the region of the poles then eventually overdamp. It is often the presence of zeros or poles, near or on the imaginary axis, which forces a restriction in the step size when following the frequency. Methods which do not have zeros or poles near the imaginary axis do better but even they must be used with restricted q to follow the frequency. Thus to correctly follow the component a small step must be employed, which suggests that the cheapest methods, ie. explicit methods, will be the most efficient.

2.3 Numerical results

To verify these predictions the 6-stage 5th order Fehlberg method, Figure 2.9, was implemented in ERK, BRK and MRK modes, to integrate the following system of ODEs,

$$\frac{dy_1}{dx} = y_2$$

$$\frac{dy_2}{dx} = y_3$$

$$\frac{dy_3}{dx} = y_4 \qquad x \in [0,10] \quad \text{(p2.1)}$$

$$\frac{dy_4}{dx} = -10001y_3 - 10000y_1$$

with initial conditions

$$y(0) = [0,6,0,-50001]^T$$

The component of interest, y_1, has the analytical solution,

$$y_1(x) = \sin(x) + 0.05\sin(100x) \qquad (2.34)$$

Hence the solution has a high frequency component, about 16Hz, superimposed upon a pure sine wave, Figure 2.11

A large fixed step was used for all modes, the step size of the MRK method being twice that of the others to compensate for the double step that it must perform, ie. it performs two steps of h/2. After only a few steps the explicit method produced spurious results. In contrast the BRK method totally damps out the high frequency and produced the pure sine wave solution, Figure 2.12. The MRK method, however, recognized the presence of the high frequency, Figure 2.13, and obtained the correct amplitude, but was unable to represent the high frequency correctly. As a result the solution produced by the MRK method is no better, in terms of accuracy than that produced by the BRK method. Table 2.5 summarises these results. The maximum global error of the first component is measured as the maximum relative difference between the analytical solution, (2.34), and the numerical one throughout the integration range. All CPU times are calculated relative to the time taken for the ERK method.

When the step is greatly reduced, all methods can follow the solution accurately. The explicit method being clearly the most efficient, Table 2.6.

Hence to follow faithfully the high frequency component an explicit method is the most efficient. It may, however, be desirable to damp out the high frequency artificially, if it is of no interest. This may often be the case in engineering applications. If so the most suitable method would be the BRK.

This desirable ability to damp out the high frequency oscillation artificially must, however, be used with caution. Consider the following system of ODEs,

$$\frac{dy_1}{dt} = y_2$$

$$\frac{dy_2}{dt} = y_3^2 - y_1$$

$$t \in [0,500] \quad \text{(p2.2)}$$

$$\frac{dy_3}{dt} = 0.5y_4$$

$$\frac{dy_4}{dt} = 4y_1 - 0.5y_3$$

with initial conditions

$$y(0) = [1,0,1,0]^T$$

This is a non-linear problem arising from the study of stellar orbits, Scheid[1983]. The first component represents the radial displacement of the orbit of the star from a reference circular orbit and y_3 the deviation of the star's orbit from the galatic plane. Astronomers solving p2.2 are interested in two energy levels, viz.

i) $E_1(t) = (y_1^2 + y_2^2)/2$

ii) $E_2(t) = (y_3^2 + y_4^2)/8$

Problem p2.2 solved by a variable step implementation of the 12-stage 8th order method of Verner[1978], with a small initial step, 1.e-5, and tight error tolerance produces the solution, for $E_2(t)$, shown in Figure 2.14, (details of the variable step algorithm employed are given in chapter 4). This agrees with the solution produced in Scheid[1983], which is characterised by a high frequency component driving a low frequency oscillation.

If the high frequency is of no interest then, by using a large initial step, it can be damped out artificially. The effect of using a large initial step, of 100, is shown in Figure 2.15. Clearly, as expected, the high frequency is damped out, but as this drives the lower frequency it too disappears. The oscillations near the end of the integration are caused by the algorithm reducing the step size to arrive exactly at the end point and in doing so detecting the oscillations.

2.4 Conclusions

In general a successful integrator of classically stiff problems should have three characteristics. Firstly they should have no poles or zeros near the negative real axis. Furthermore they must have

$$\arg(E(q)) = \arg(\exp(q)) = 0 \qquad (2.35)$$

for q real and negative and

$$|E(q)| \approx |\exp(q)| \qquad (2.36)$$

(in the absolute sense), for Re(q) less than zero.

For the highly oscillatory case it has been shown that

$$|E(q)| = |\exp(q)| = 1 \qquad (2.37)$$

can be achieved on the whole of the imaginary plane, but in none of the cases considered has

$$\arg(E(q)) = \arg(\exp(q)) \qquad (2.38)$$

been achieved.

A successful integrator will have no poles or zeros close to the imaginary axis. When integrating these types of problems (highly oscillatory) neither A-stability or precise A-stability is necessary. On balance explicit methods will be most efficient for these problems.

$1/2 + \sqrt{3}/6$	$1/4$	$1/4 + \sqrt{3}/6$
$1/2 - \sqrt{3}/6$	$1/4 - \sqrt{3}/6$	$1/4$
	$1/2$	$1/2$

Table 2.1 : Butcher matrix of 2-stage 4th order Runge-Kutta method

Order	1	2	3	4	5	6
BRK	90.0°	90.0°	88.2°	83.9°	79.1°	74.3°
BDF	90.0°	90.0°	88.0°	73.0°	51.0°	18.0°

Table 2.2 : Comparison of A(α)-stability region for BRK and BDF.

Order	Position of zeros
1	-1.0
2	-0.5 ± 0.5i
3	-0.1867309 ± 0.4807739i -0.6265383
4	-0.0426266 ± 0.3946330i -0.4573733 ± 0.2351005i

Table 2.3 : Location of zeros of s-stage s order ERK methods

Order	Position of zeros
5	-0.4243994
	0.02526698 ± 0.2966870i
	-0.2714430 ± 0.2825228i
	-0.08324843

Table 2.4 : Location of zeros of 6-stage 5th order ERK method

Mode	Step size	Max. Global error y_1	Relative CPU time
ERK	0.1	8.00 e287	1.00
BRK	0.1	4.99 e-2	1.87
MRK	0.2	9.58 e-2	1.60

Table 2.5 : Comparison of the 5th order method of Fehlberg in different modes (large step)

Mode	Step size	Max. Global error y_1	Relative CPU time
ERK	0.01	4.04 e-2	1.00
BRK	0.01	2.49 e-2	2.10
MRK	0.02	1.92 e-2	1.42

Table 2.6 : Comparison of the 5th order method of Fehlberg in different modes (small step)

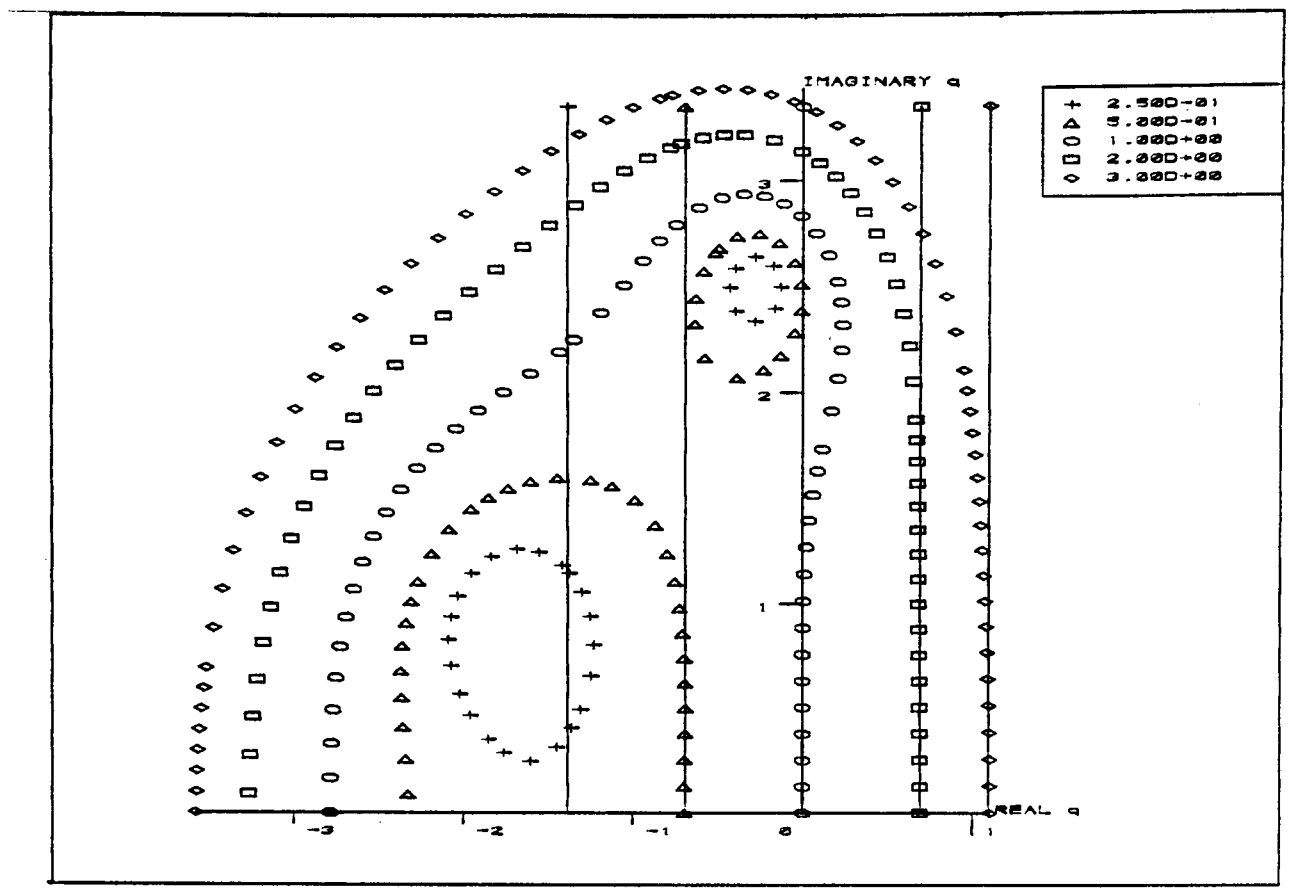


Figure 2.1a : Modulus plot of Padé $R_{4,0}$ approximation.

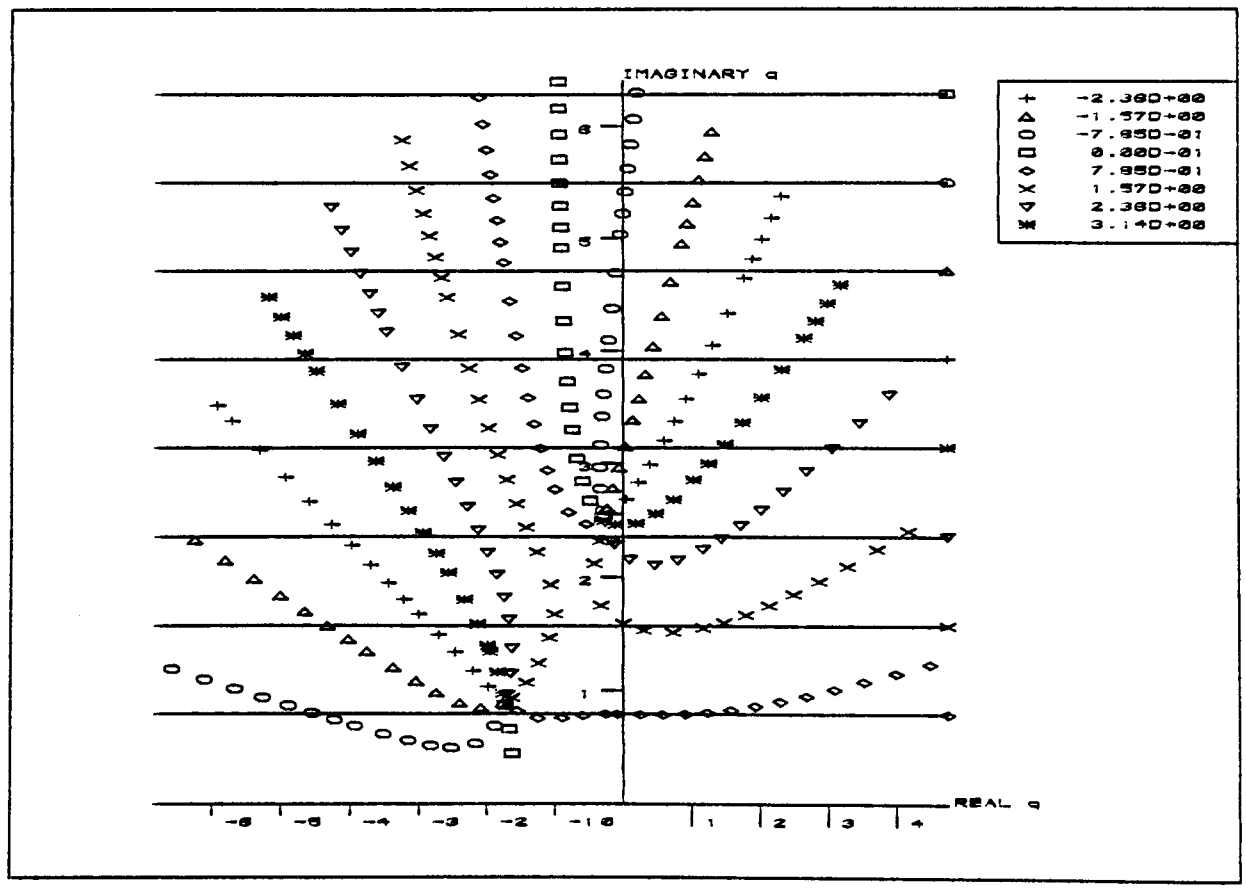


Figure 2.1b : Argument plot of Padé $R_{4,0}$ approximation.

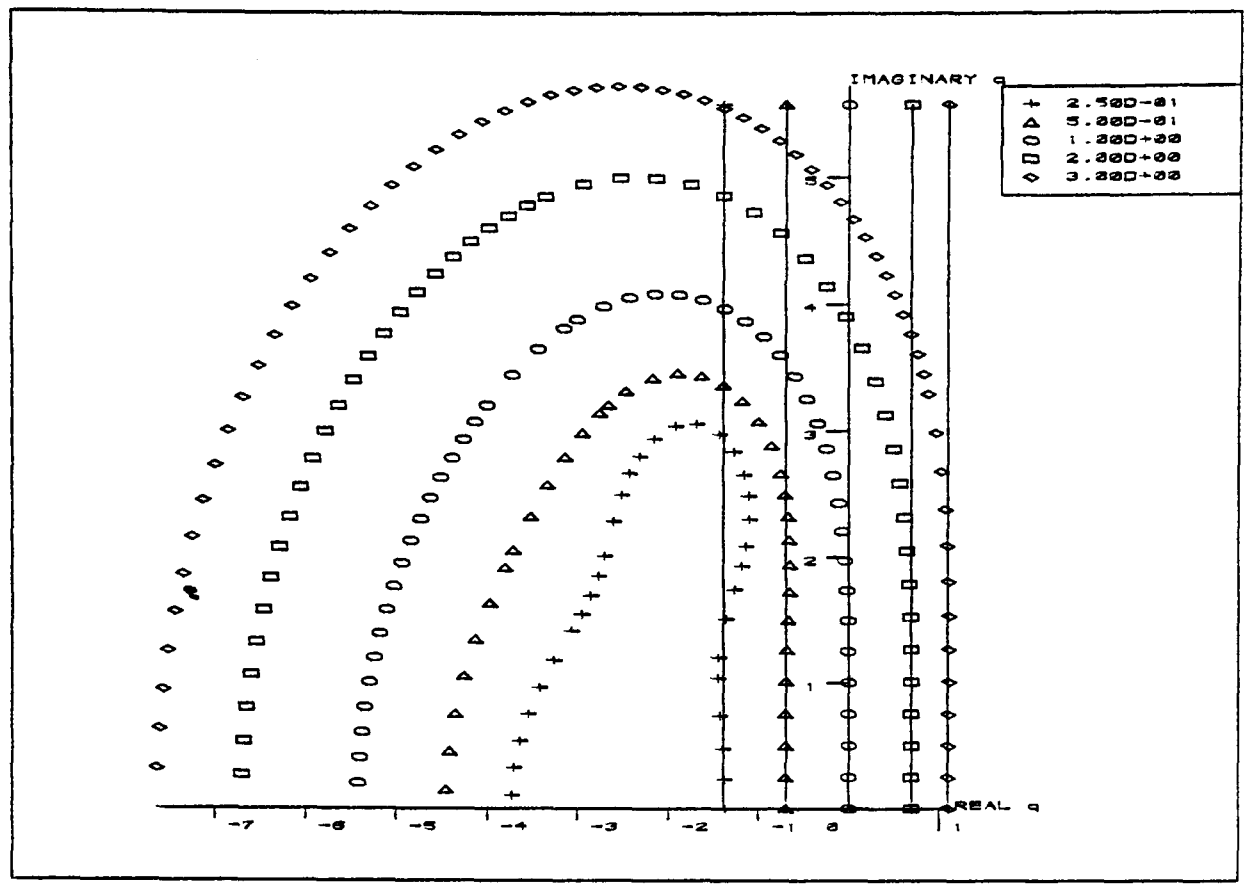


Figure 2.2a : Modulus plot of Padé $R_{3,1}$ approximation.

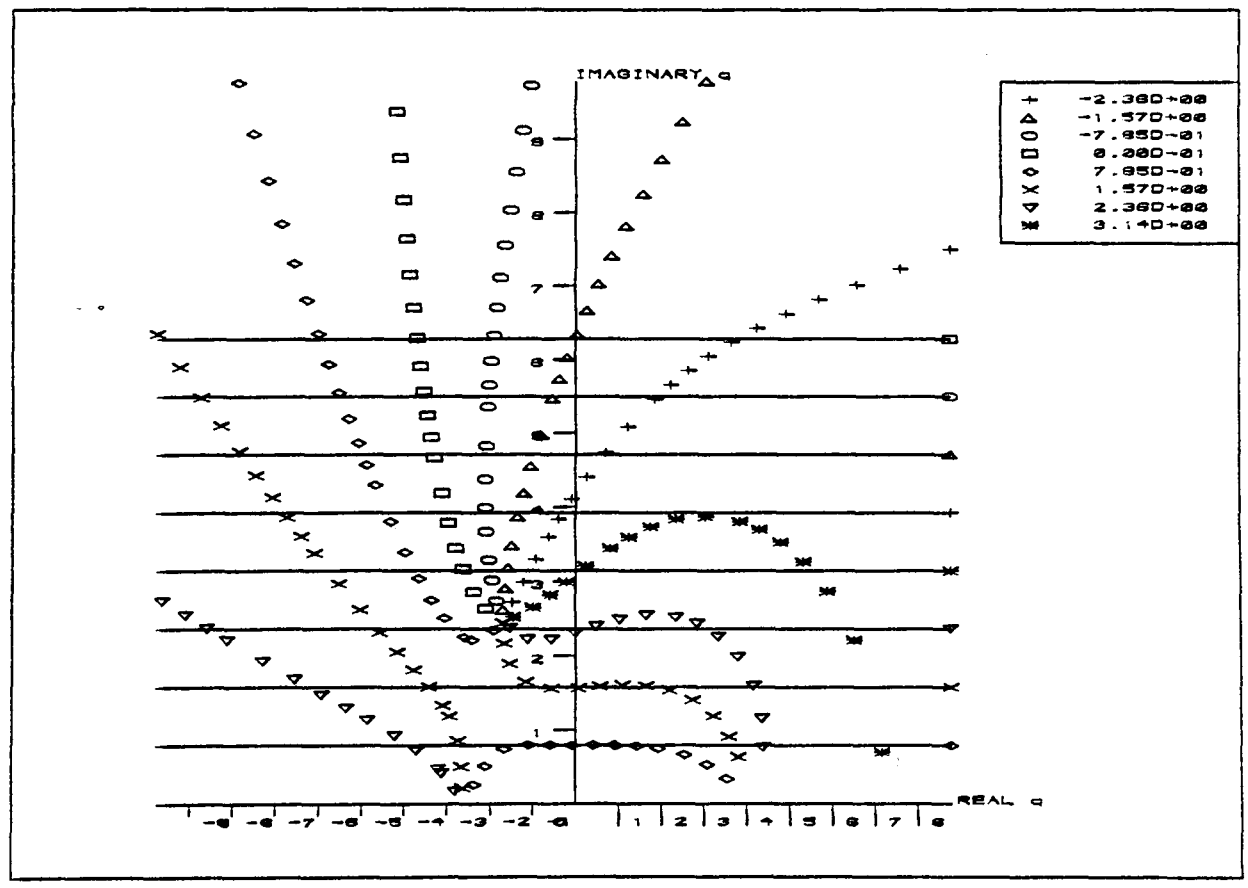


Figure 2.2b : Argument plot of Padé $R_{3,1}$ approximation.

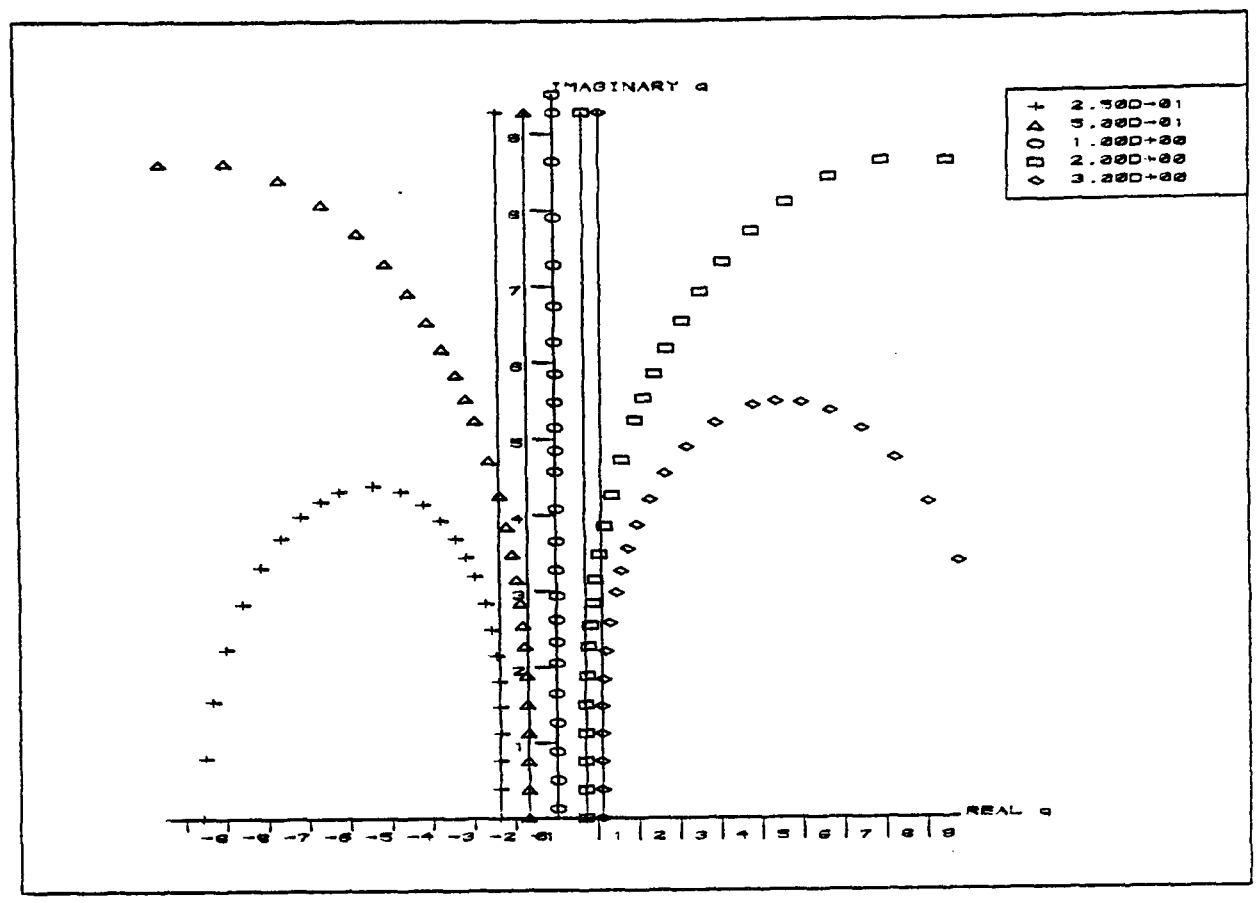


Figure 2.3a : Modulus plot of Padé $R_{2,2}$ approximation.

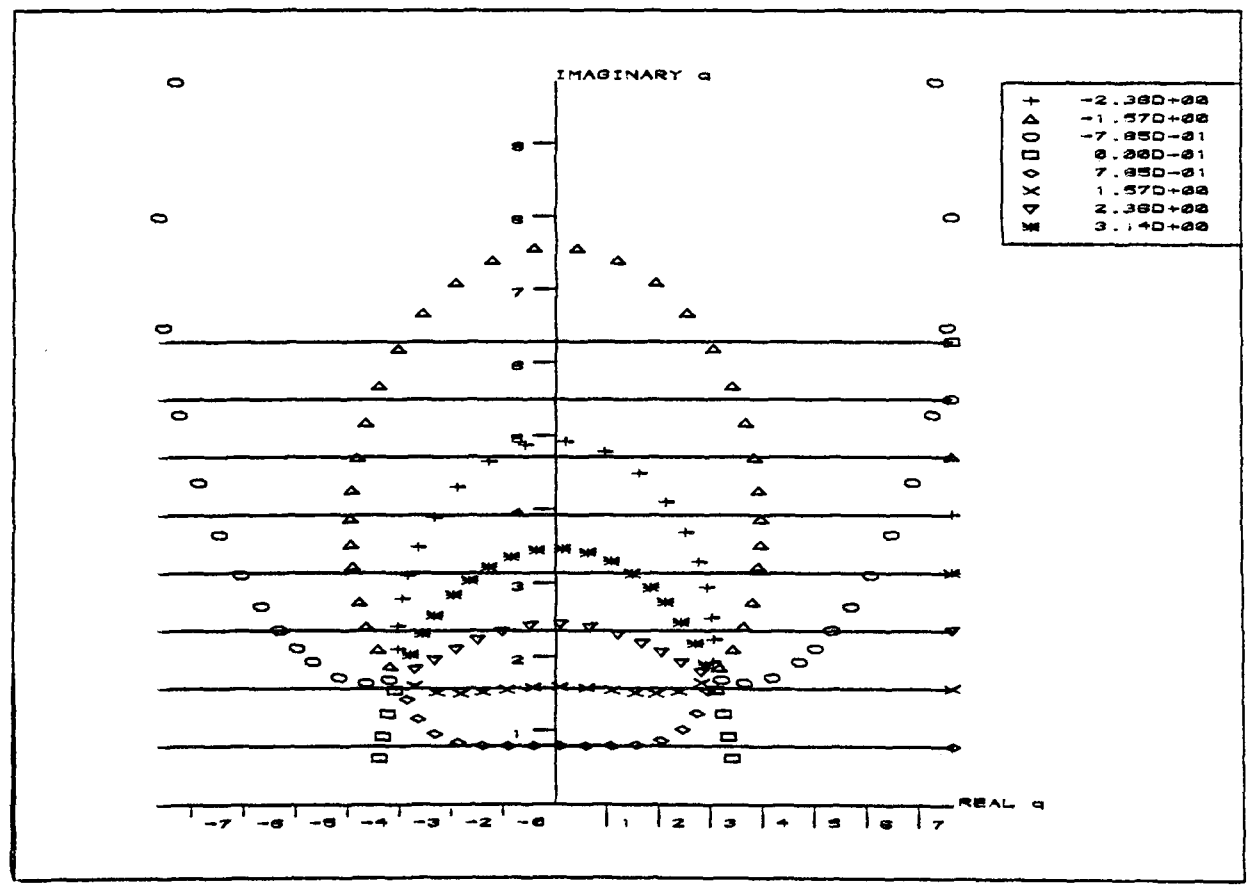


Figure 2.3b : Argument plot of Padé $R_{2,2}$ approximation.

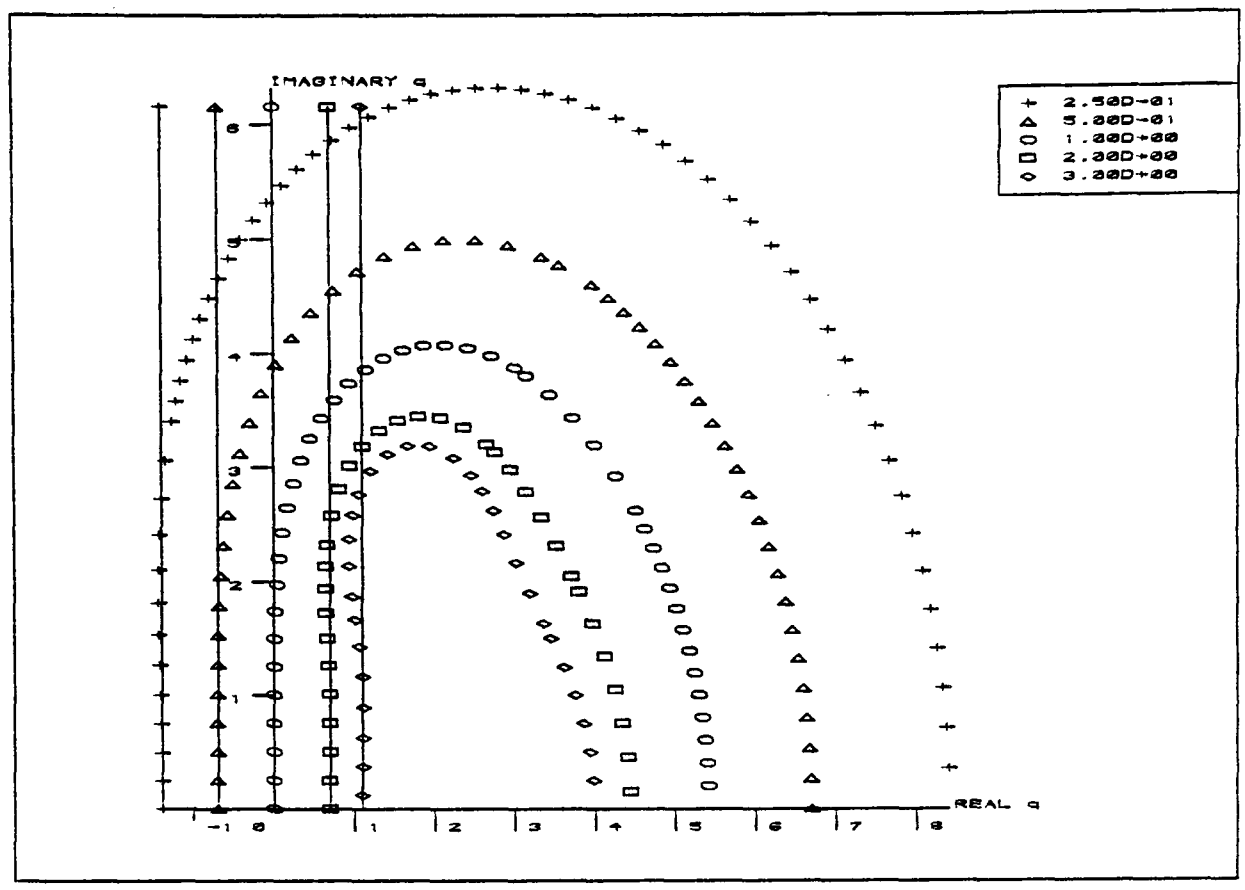


Figure 2.4a : Modulus plot of Padé $R_{1,3}$ approximation.

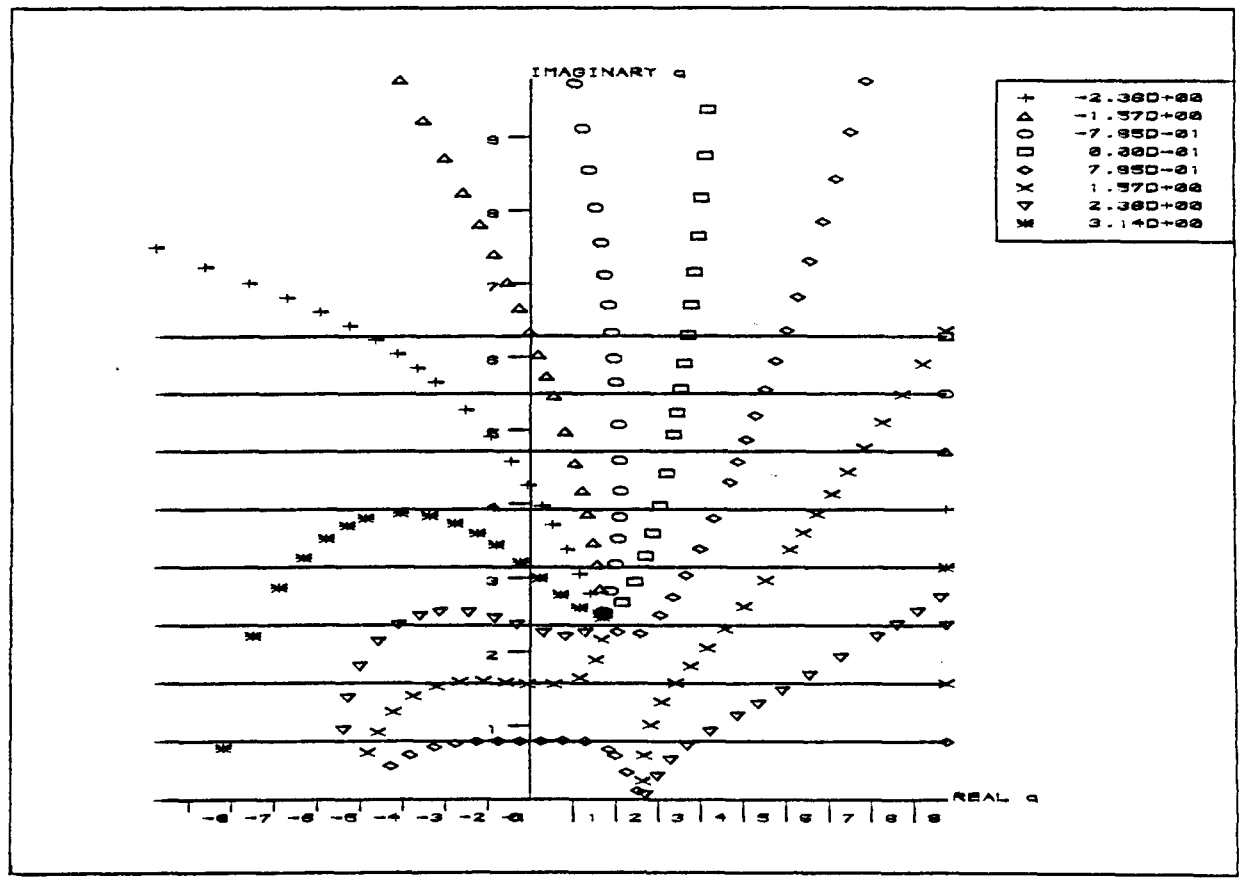


Figure 2.4b : Argument plot of Padé $R_{1,3}$ approximation.

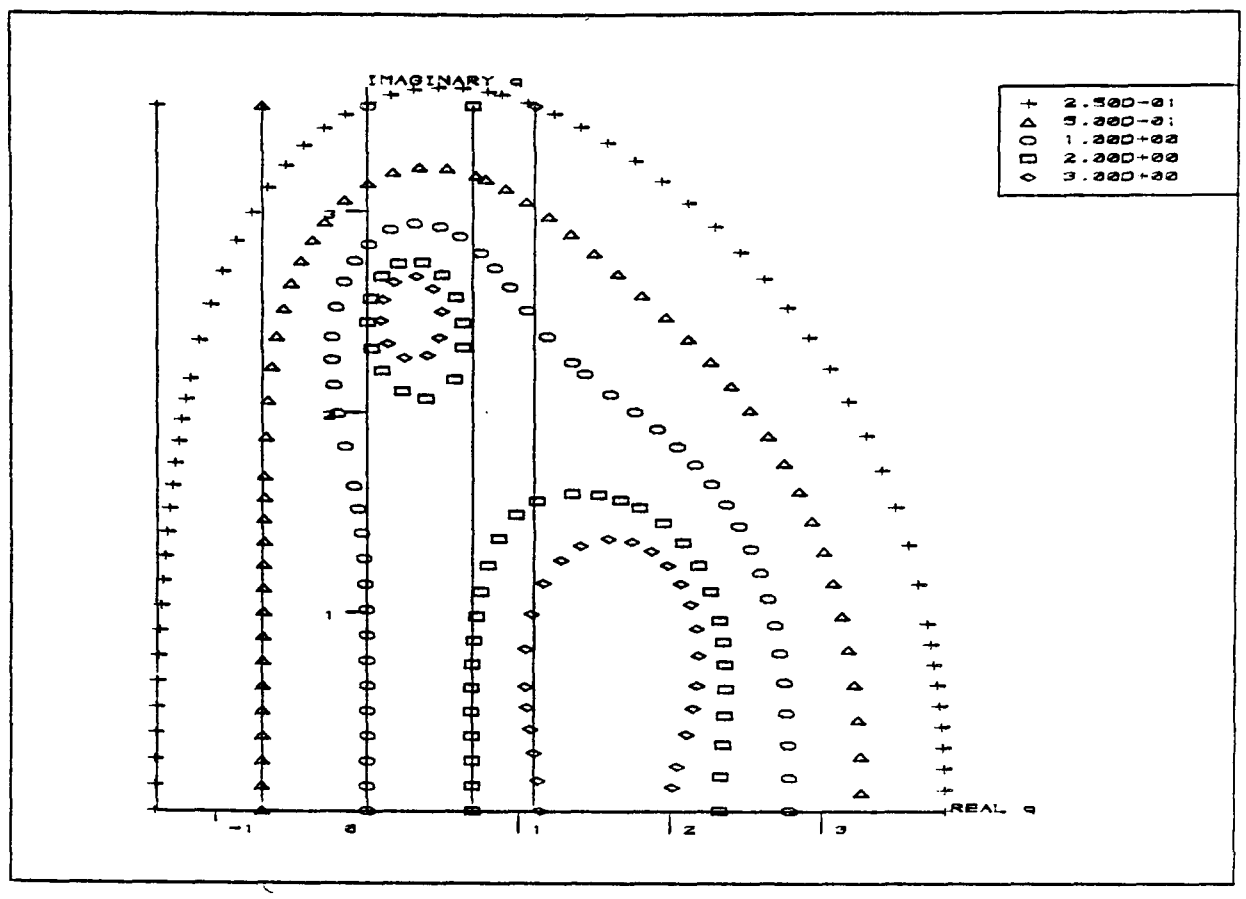


Figure 2.5a : Modulus plot of Padé $R_{0,4}$ approximation.

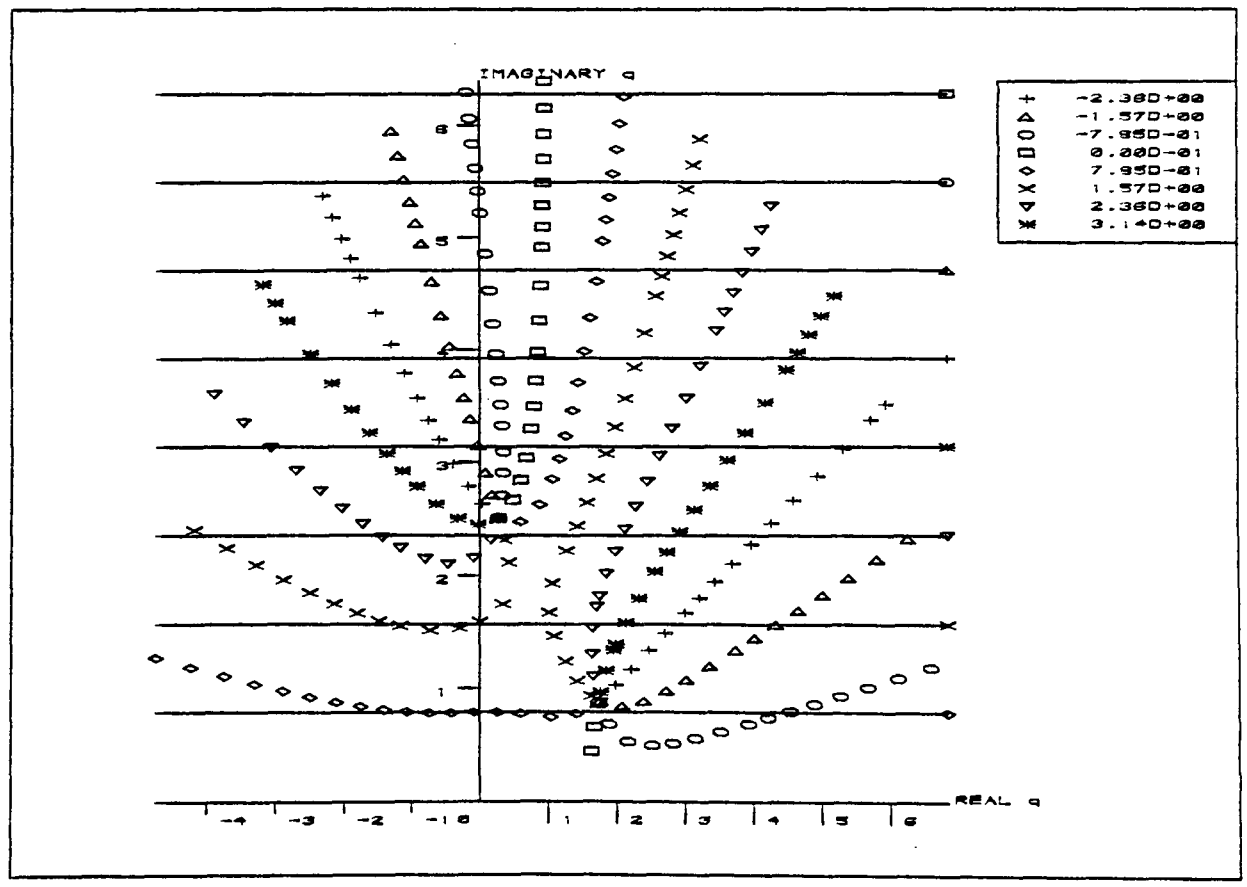


Figure 2.5b : Argument plot of Padé $R_{0,4}$ approximation.

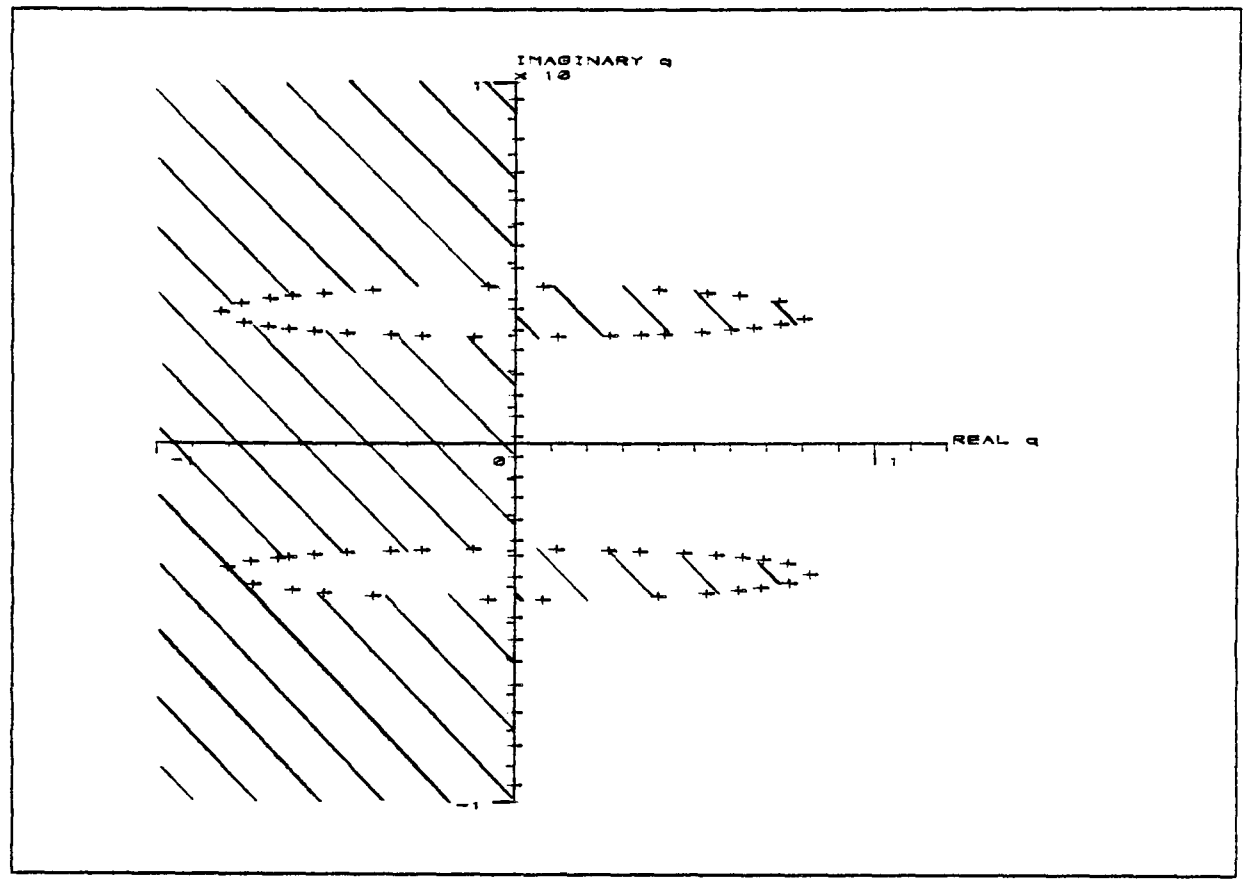


Figure 2.6 : Absolute stability region of Fehlberg's 5th order method in MRK mode.

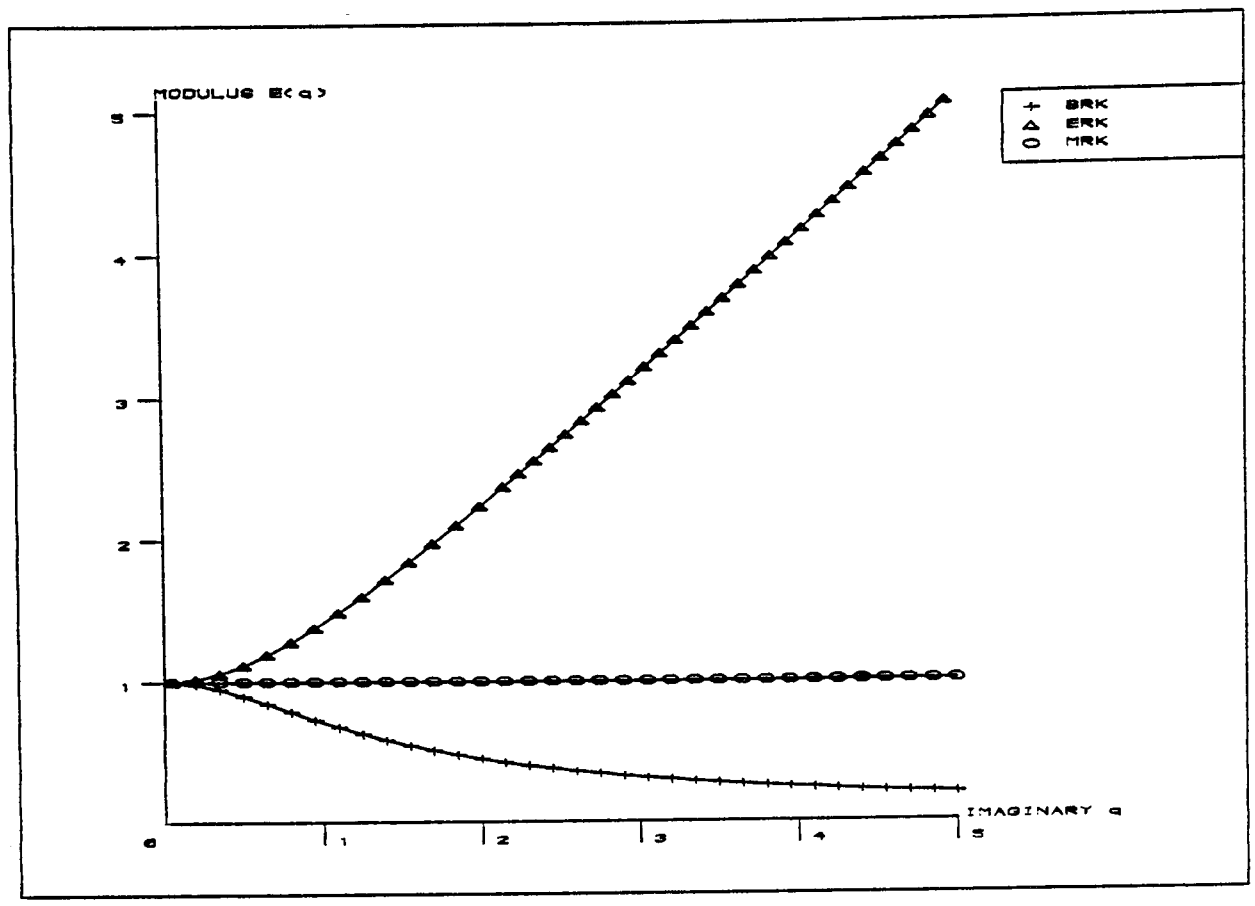


Figure 2.7a : Modulus plot of Euler based methods.

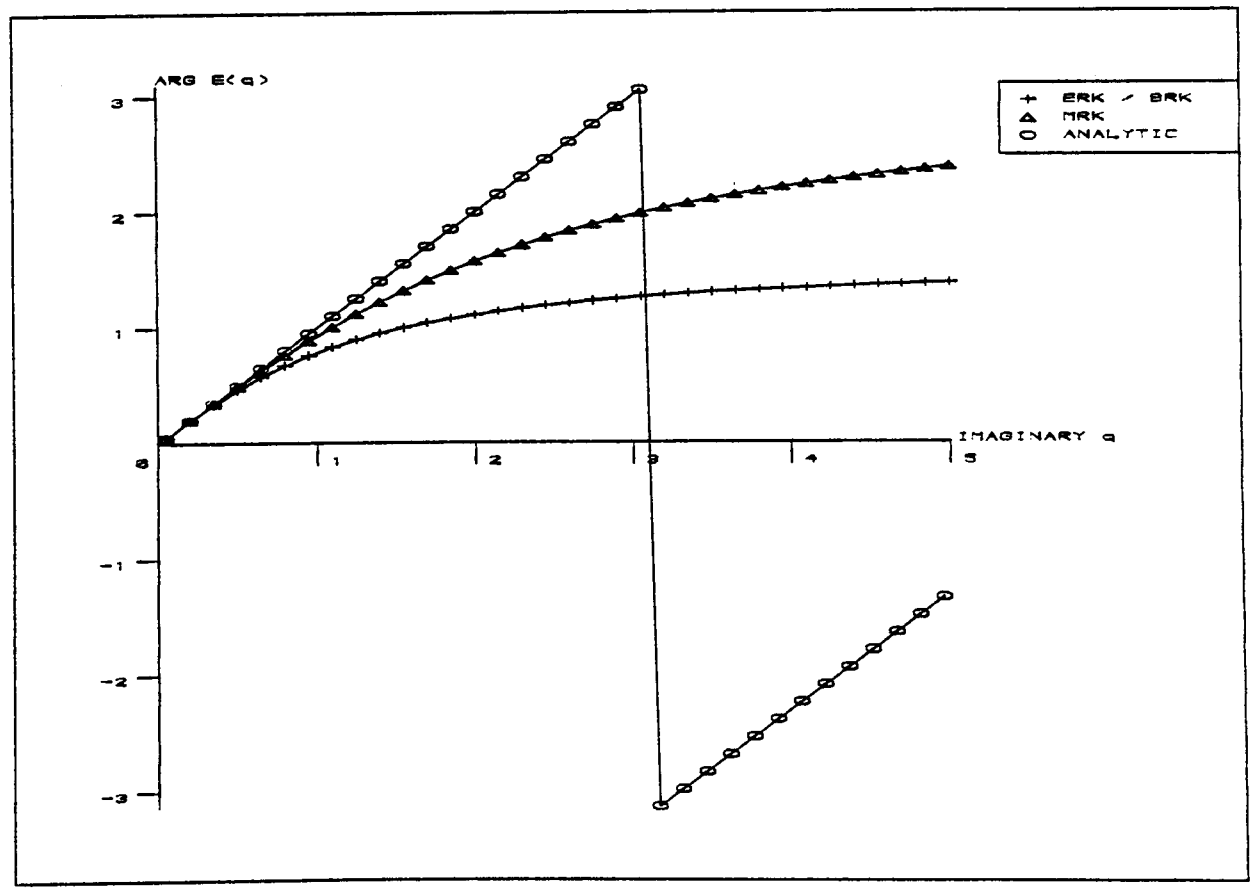


Figure 2.7b : Argument plot of Euler based methods.

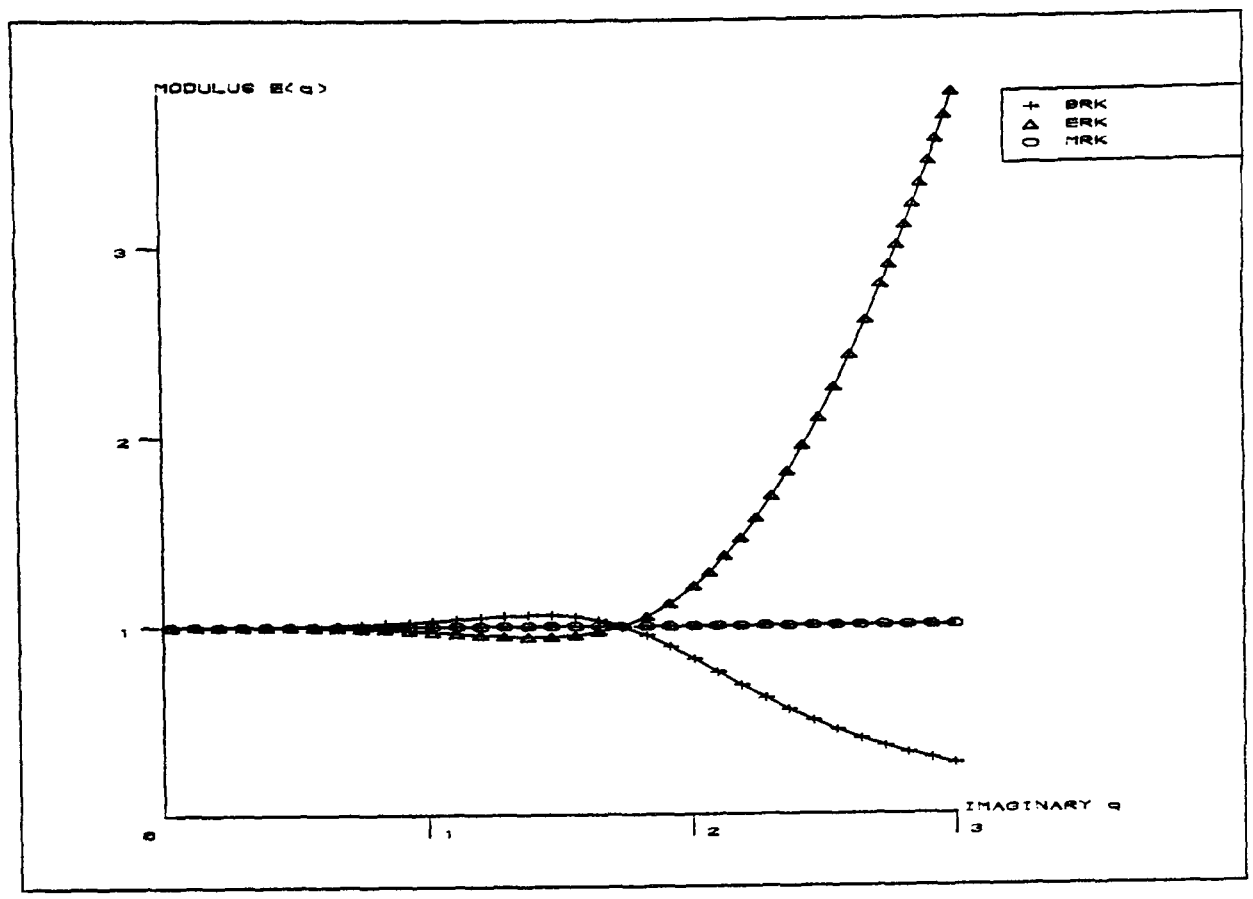


Figure 2.8a : Modulus plot of 3rd order based methods

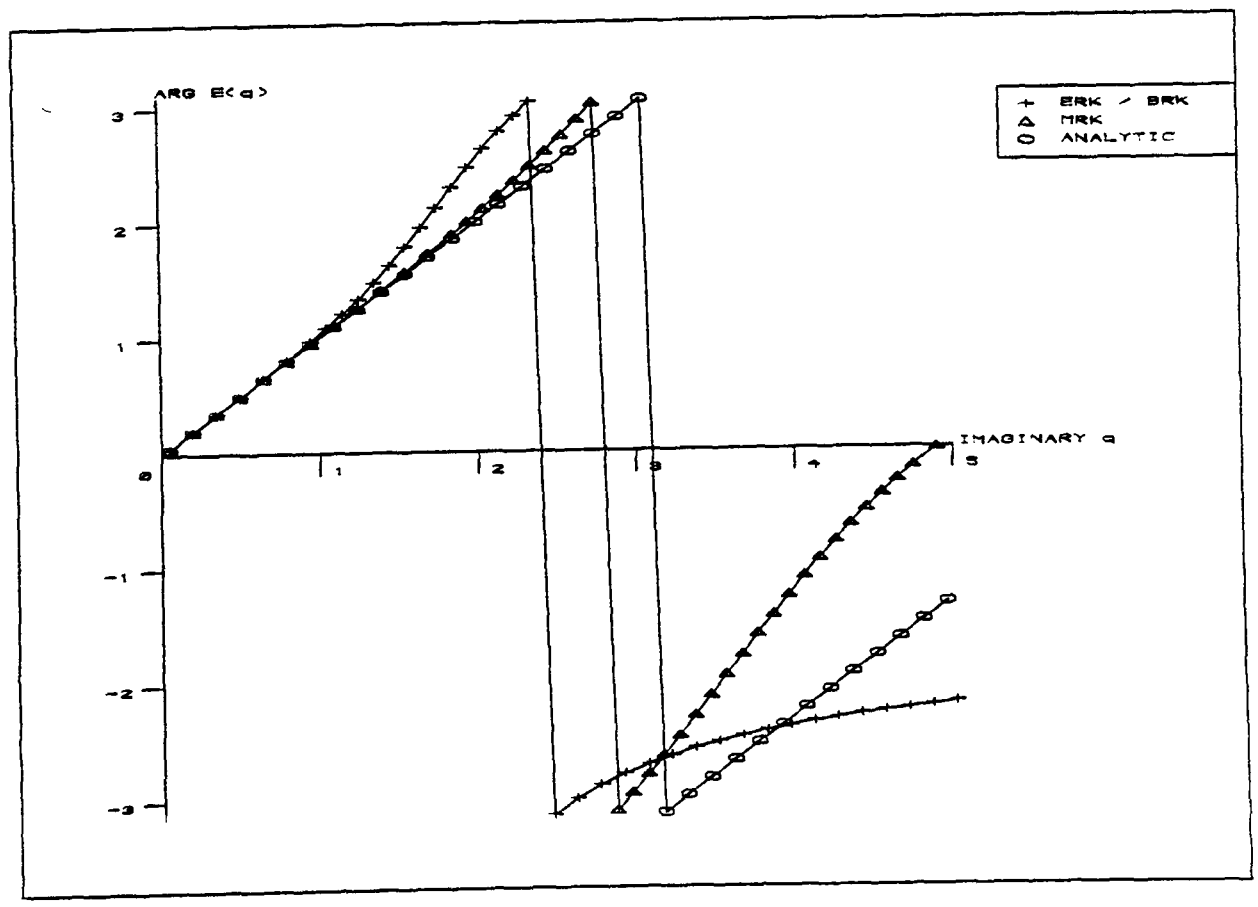


Figure 2.8b : Argument plot of 3rd order based methods.

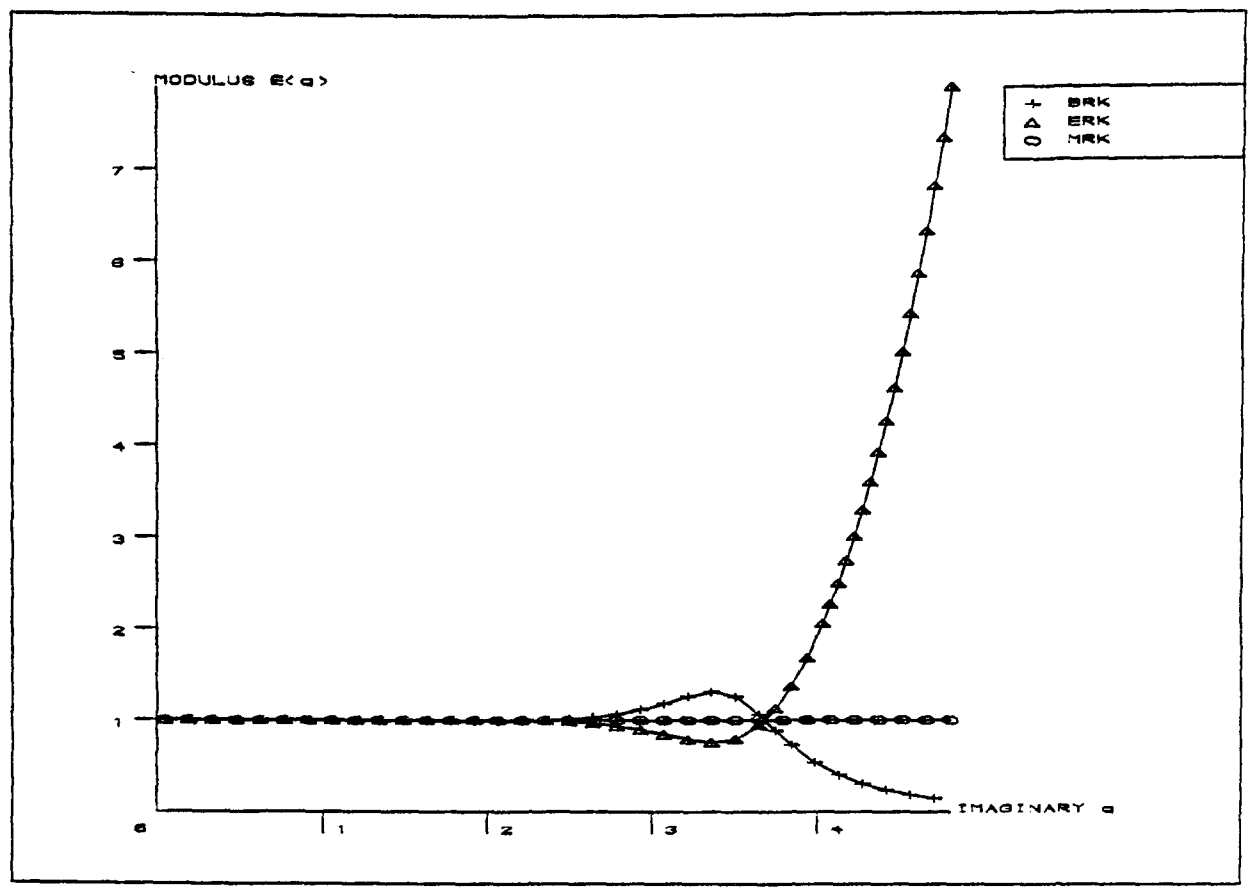


Figure 2.9a : Modulus plot of 5th order based methods.

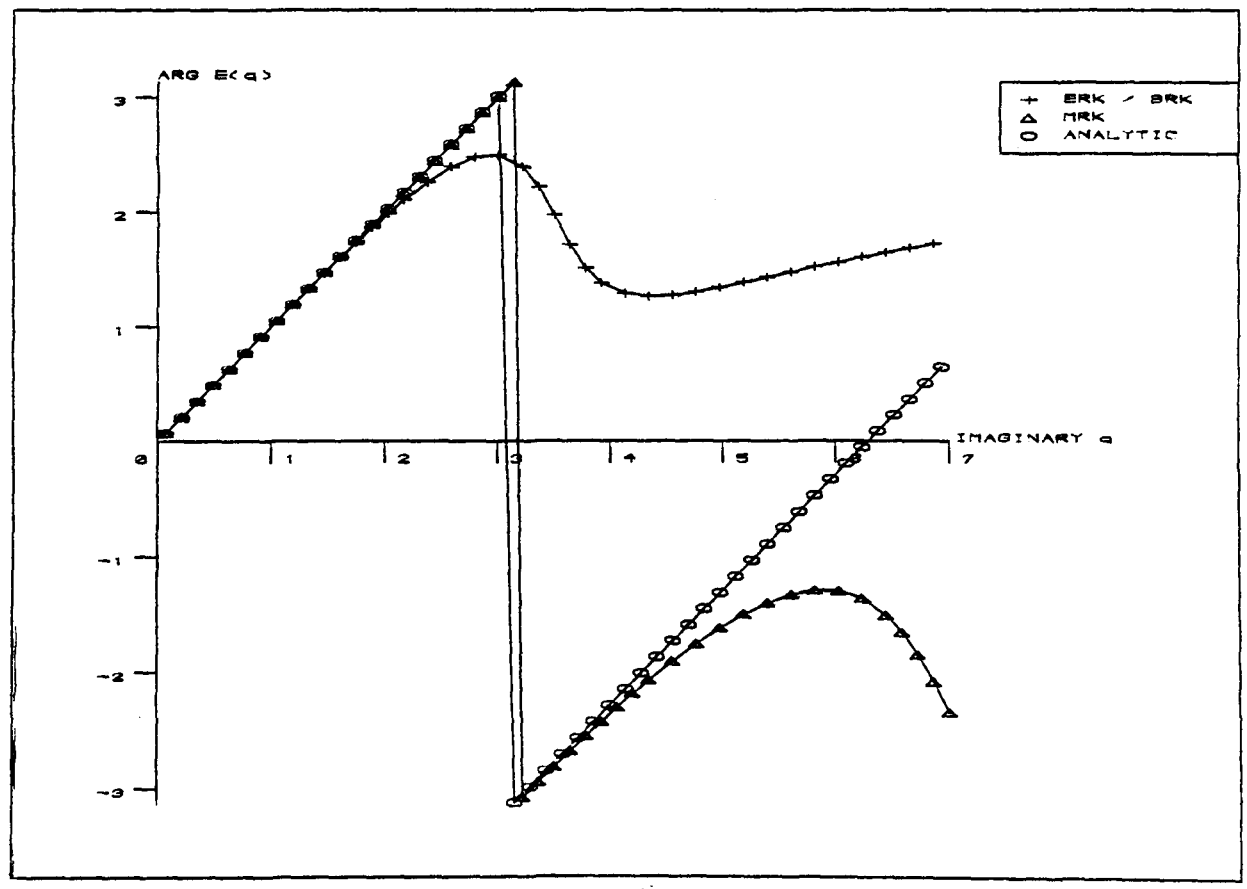


Figure 2.9b : Argument plot of 5th order based methods.

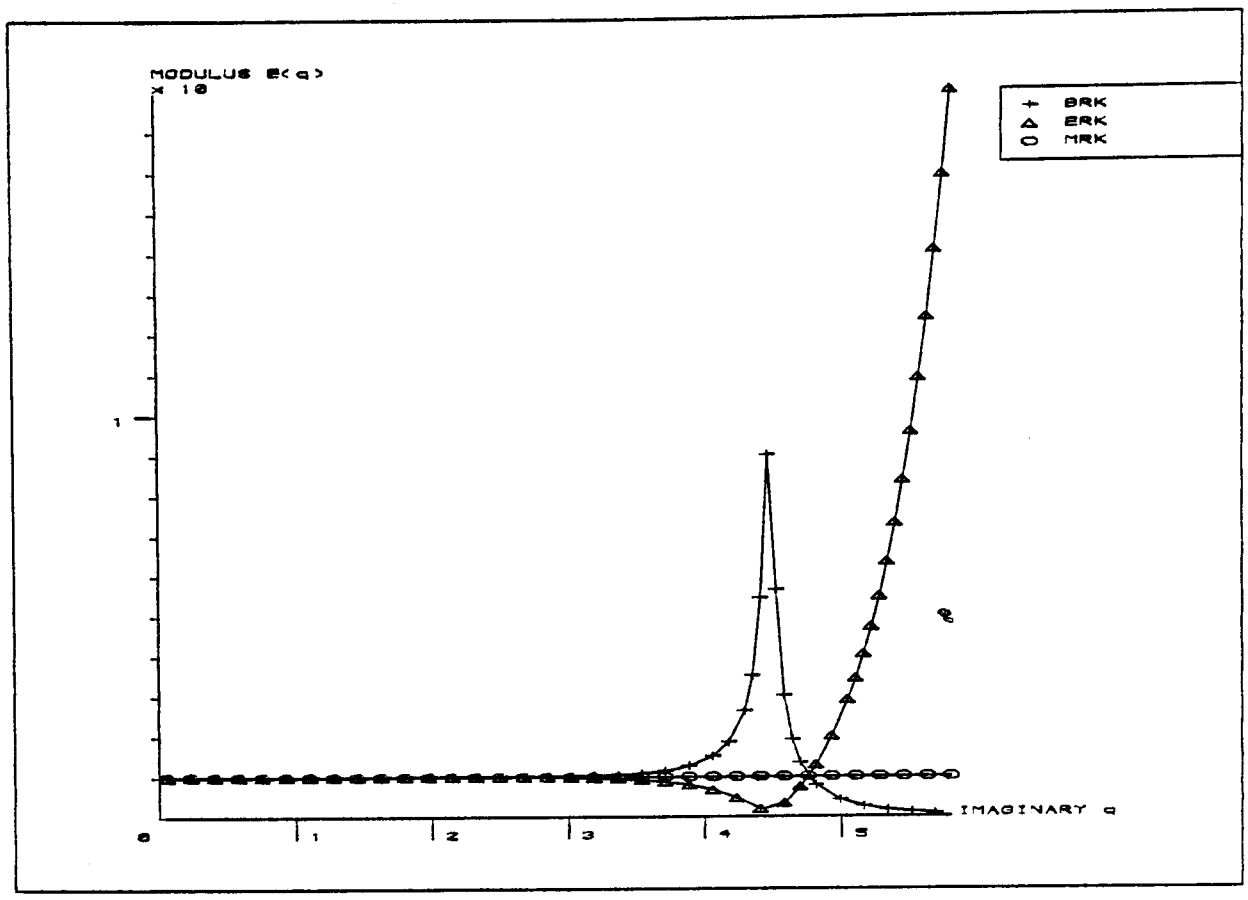


Figure 2.10a : Modulus plot of 8th order based methods.

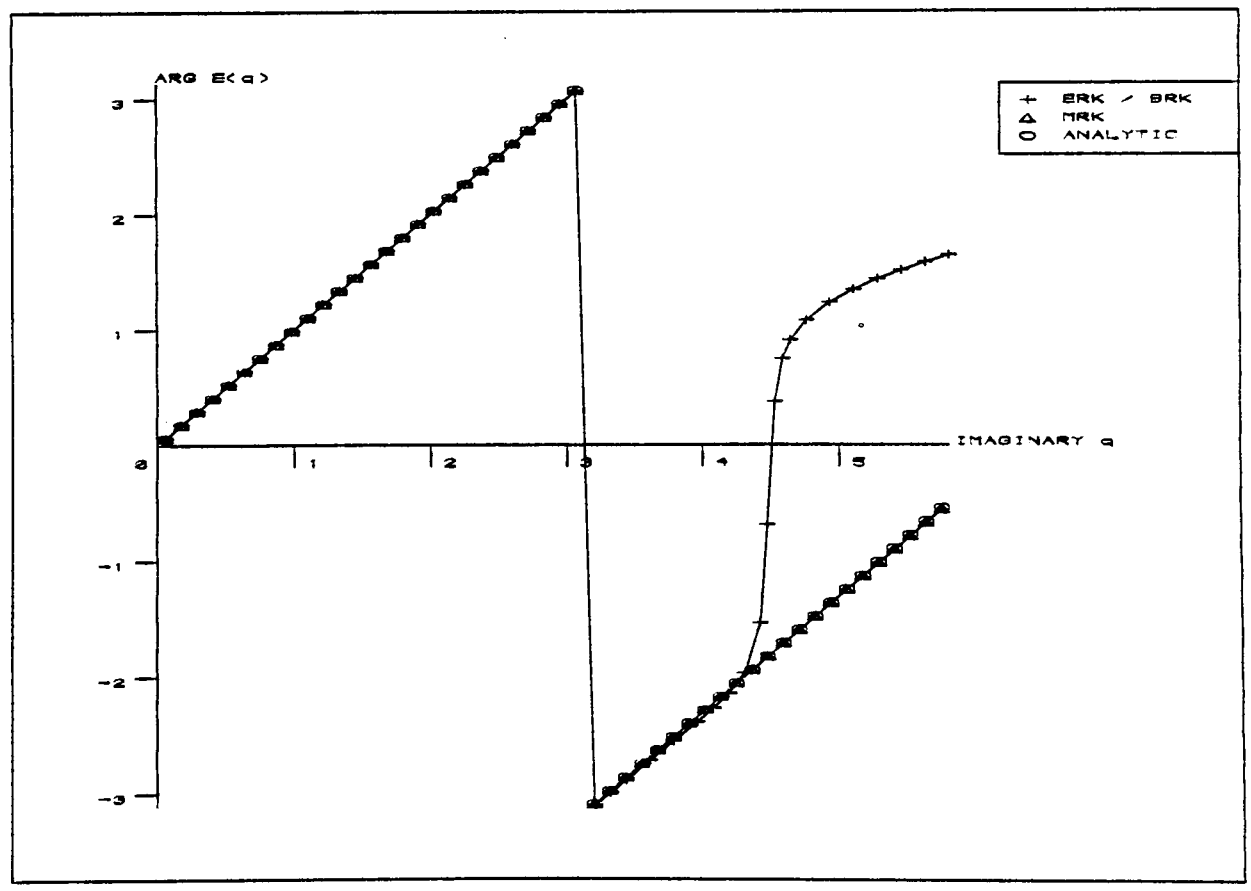


Figure 2.10b : Argument plot of 8th order based methods.

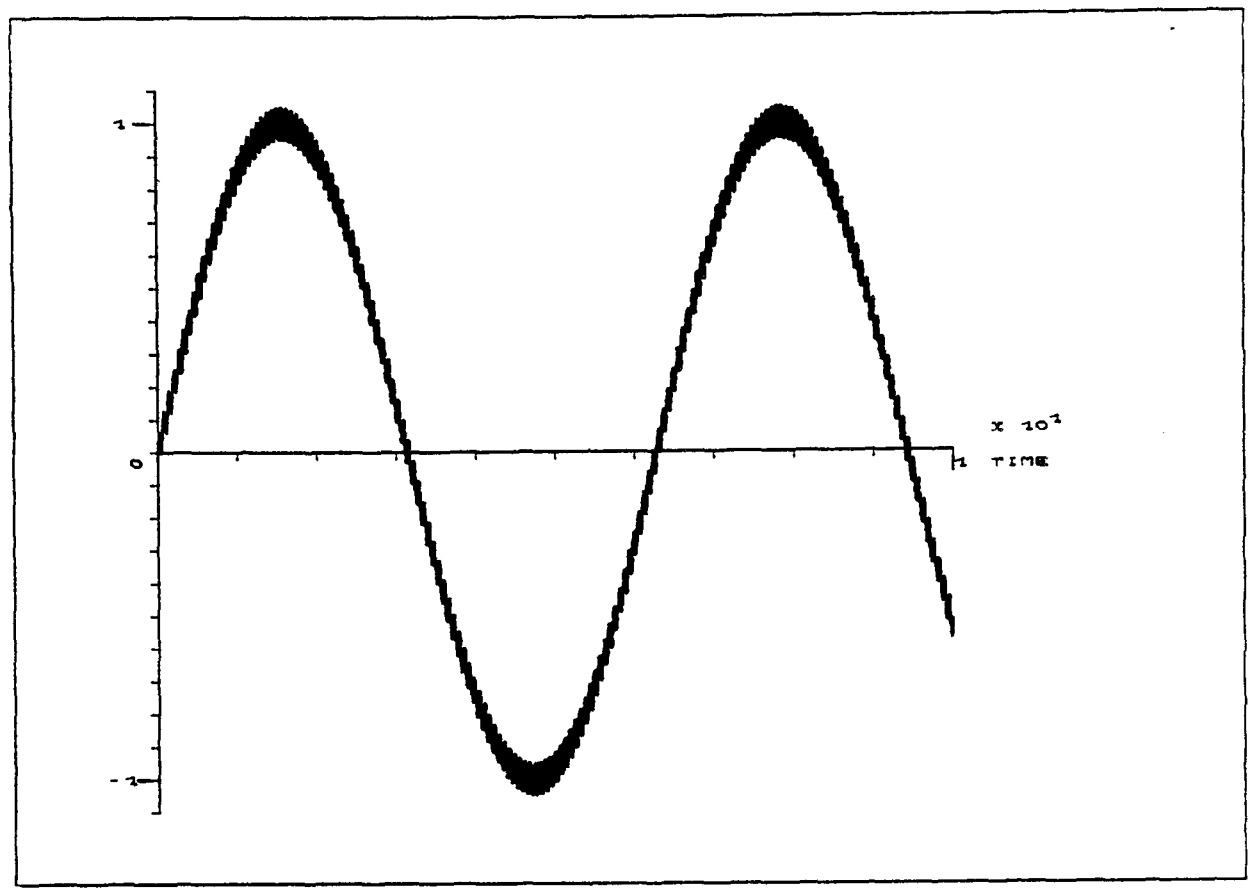


Figure 2.11 : Analytical solution of $y_1(x)$ for problem p2.1.

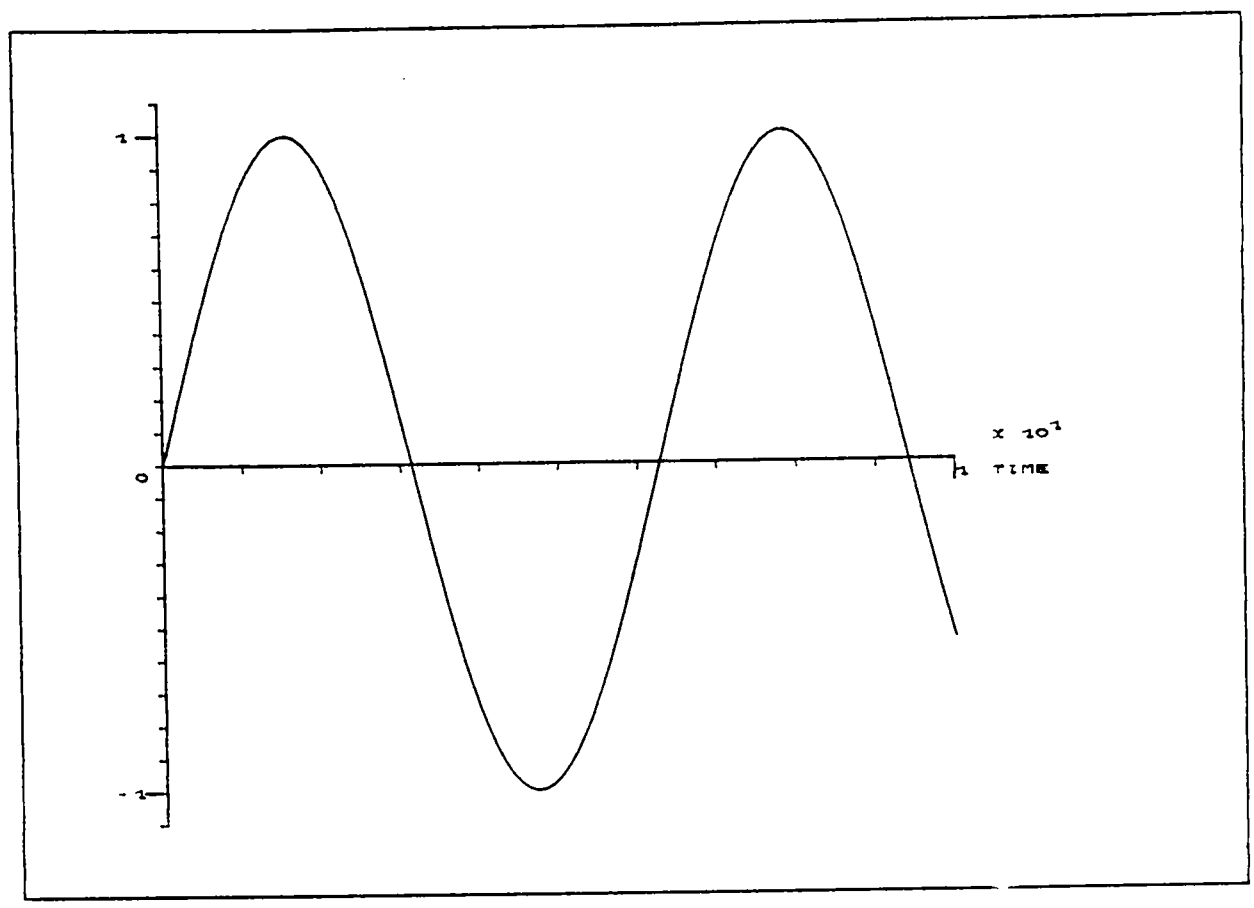


Figure 2.12 : Solution of y_1 for p2.1 by BRK method.

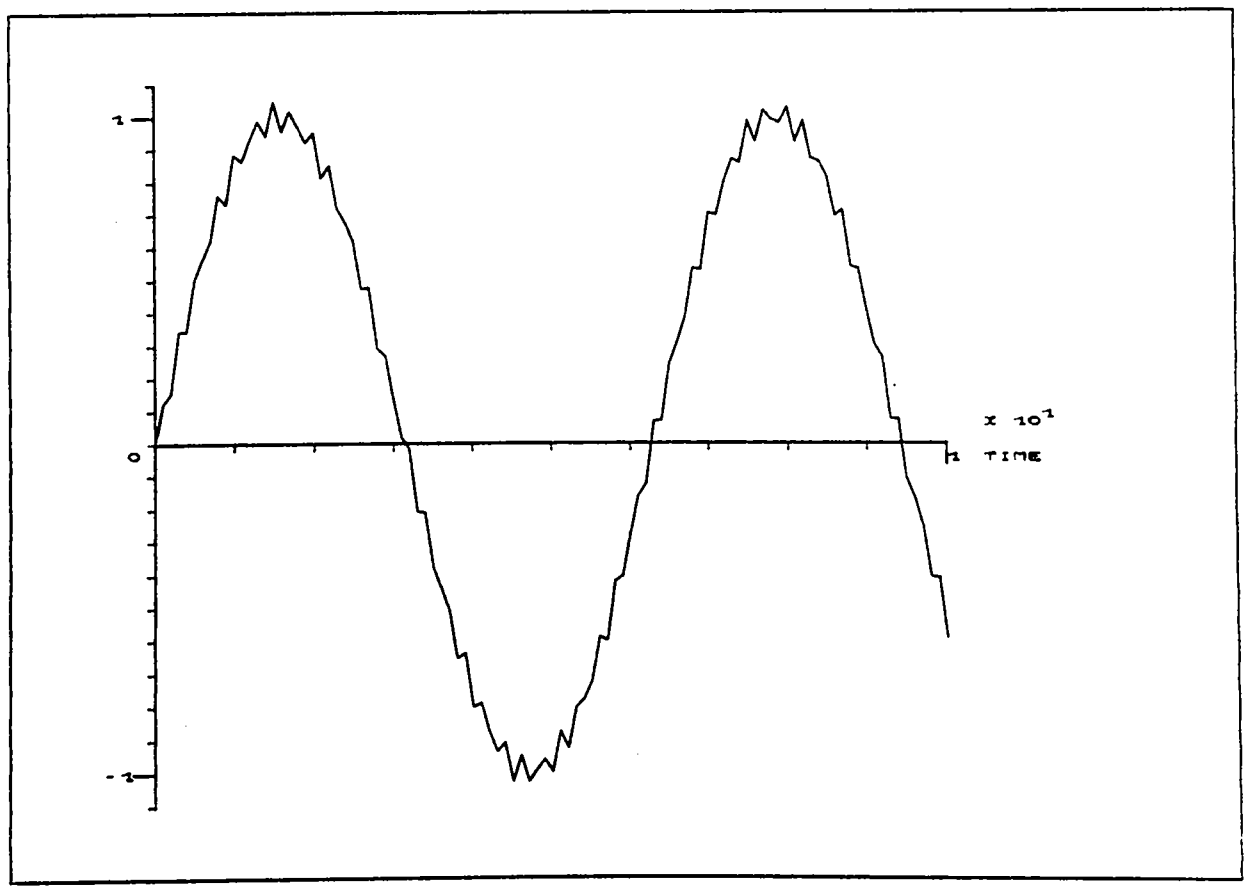


Figure 2.13 : Solution of y_1 for p2.1 by MRK method.

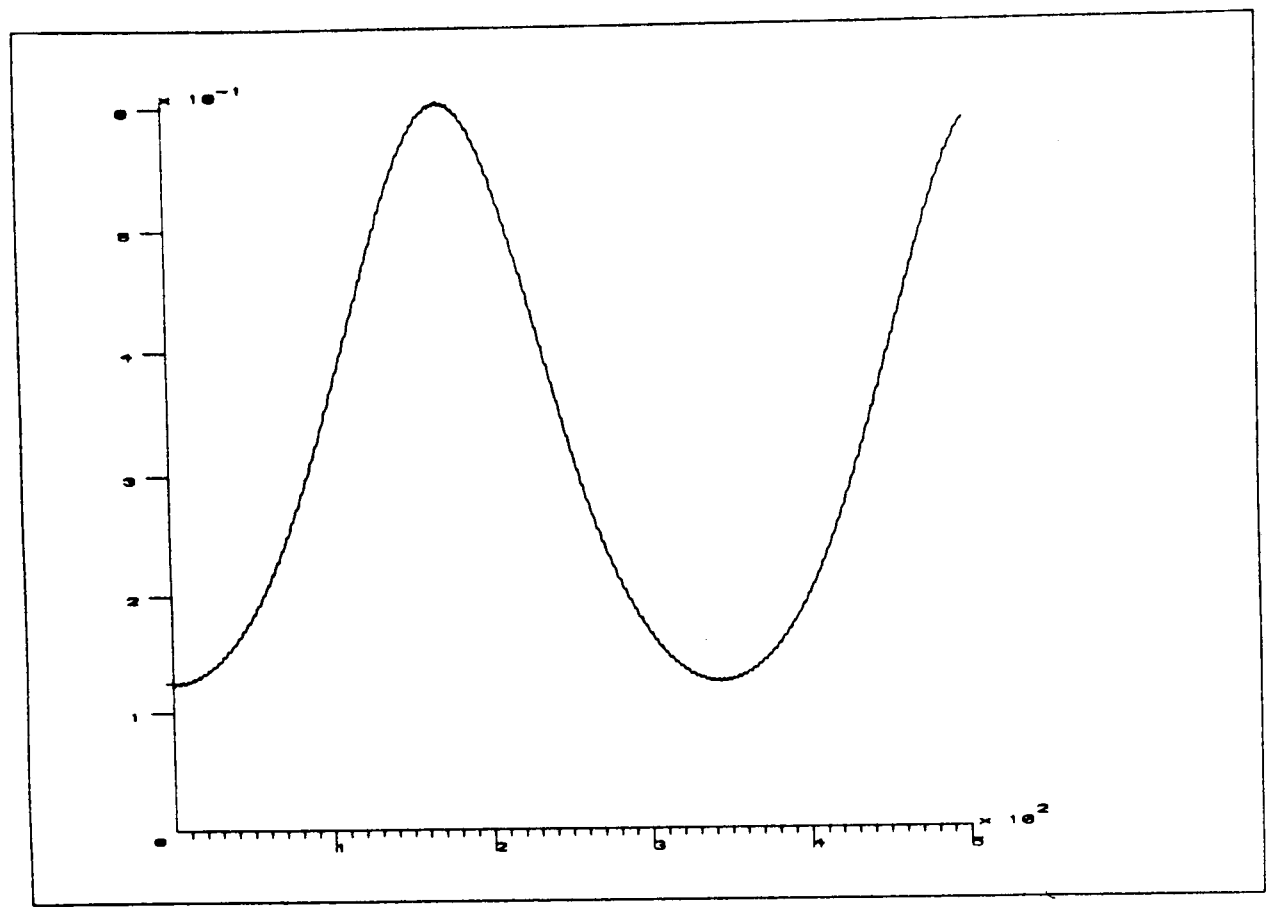


Figure 2.14 : Energy level $E_2(t)$ for p2.2, small initial step.

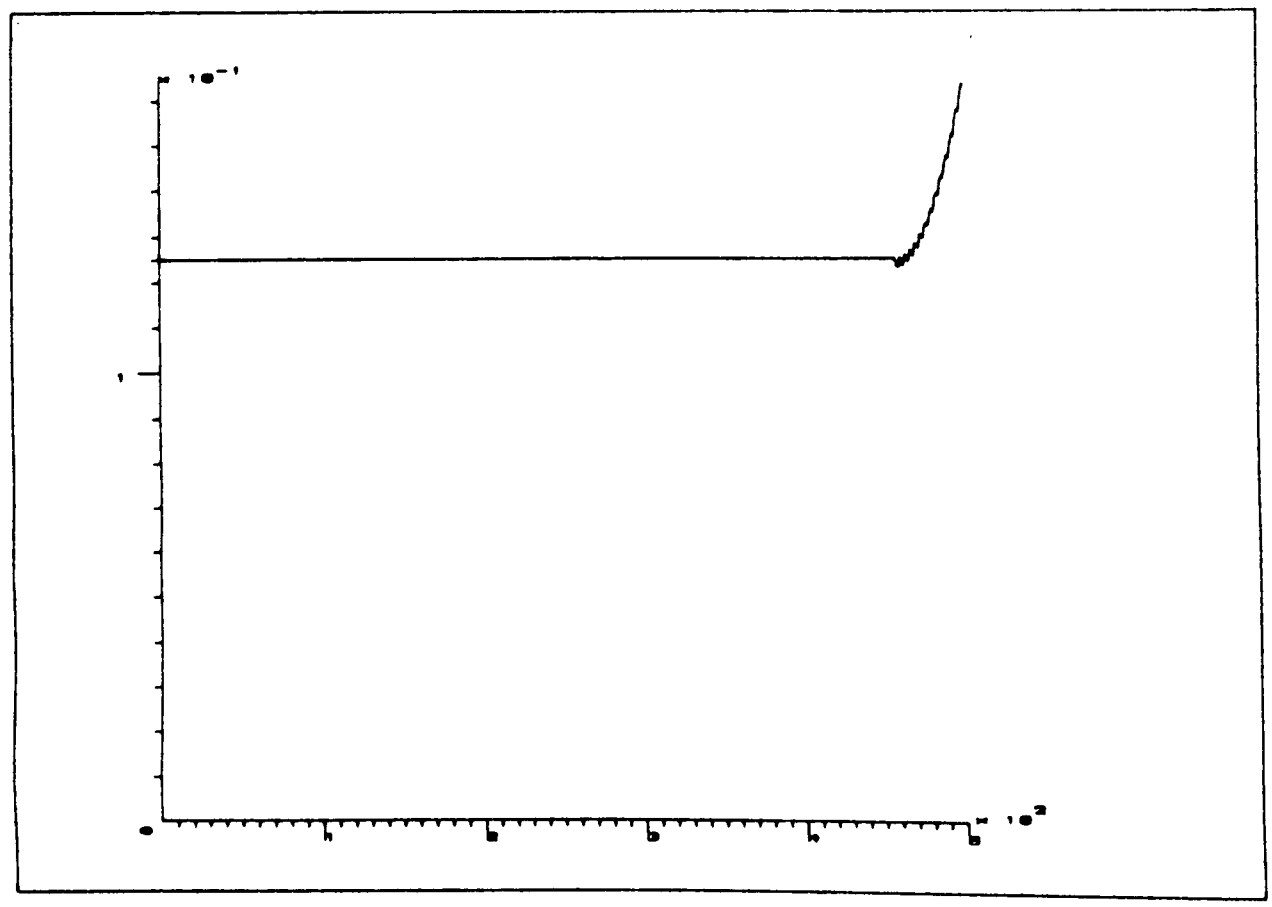


Figure 2.15 : Energy level $E_2(t)$ for problem p2.2, large initial step

Chapter 3 : BACKWARD RUNGE-KUTTA METHODS

This chapter discusses the basic characteristics of Backward Runge-Kutta methods for the numerical integration of stiff systems of ODEs. In particular the superior damping properties of BRK methods over linear multistep methods will be discussed. The close coupling of BRK methods with ERK methods will be emphasized and further absolute stability regions together with the new modulus and argument plots presented.

The implementation of fixed step BRK methods into a computer code is discussed and numerical results presented for a wide range of problems. These results show the enormous potential of the method for solving many stiff systems.

3.1 Derivation of Backward Runge-Kutta methods

By considering the MIRK methods of Cash[1975], (1.23), and setting r = 0 then an s-stage class of MIRK method given by,

$$y_{n+1} = y_n + h \sum_{i=1}^{s} c_i k_i$$

$$k_i = f(x_{n+1} - hb_i,\ y_{n+1} - h \sum_{j=1}^{i-1} a_{ij} k_j) \quad i=1(1)s \qquad (3.1)$$

is generated. This formula can be considered as an explicit Runge-Kutta method with step size -h moving backwards from y_{n+1} to y_n. Clearly any coefficients from an ERK method can be used to form the corresponding BRK method. For example consider the 1-stage 1st order Euler method,

$$y_{n+1} = y_n + hk_1$$

$$k_1 = f(x_n, y_n) \qquad (3.2)$$

The corresponding BRK is constructed by integrating backwards ie. replacing x_n by x_{n+1} and h by -h, in (3.2). This leads to the

following method,

$$y_{n+1} = y_n + hk_1$$
$$k_1 = f(x_{n+1}, y_{n+1}) \qquad (3.3)$$

which is the 1-stage 1st order implicit method, Backward Euler.

Expressing (3.2) as a Butcher matrix, leads to

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array} \qquad \begin{array}{c|c} b & A \\ \hline & c^T \end{array} \qquad (3.4)$$

From the Butcher matrix of an ERK method the Butcher matrix of the corresponding BRK method can be derived by applying the following transformation,

$$\begin{array}{c|c} u-b & C - A \\ \hline & c^T \end{array} \qquad (3.5)$$

where C is a s×s matrix with all rows comprising c^T and u is the s-component vector $(1, \ldots, 1)^T$. Applying this transformation to (3.4) yields,

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array} \qquad (3.6)$$

the Butcher matrix of Backward Euler. Thus BRK methods are fully implicit methods but implemented as MIRK methods, ie. implicit in y_{n+1} only.

3.2 Order of BRK methods

When a BRK method is formed from an ERK method the coefficients remain the same and hence the same order constraint equations must be upheld. Thus the order of a BRK method is the same as that of the corresponding ERK method.

3.3 Absolute Stability regions of BRK methods

Consider the general 3-stage BRK method,

$$y_{n+1} = y_n + h[c_1k_1 + c_2k_2 + c_3k_3] \tag{3.7}$$

where the k values are computed as

$$\begin{aligned} k_1 &= f(x_{n+1}, y_{n+1}) \\ k_2 &= f(x_{n+1} - hb_2,\ y_{n+1} - ha_{21}k_1) \\ k_3 &= f(x_{n+1} - hb_3,\ y_{n+1} - ha_{31}k_1 - ha_{32}k_2) \end{aligned} \tag{3.8}$$

applied to the standard test problem, (2.1) with constant step h,

$$\begin{aligned} k_1 &= \lambda y_{n+1} \\ k_2 &= \lambda(y_{n+1} - ha_{21}k_1) \\ &= \lambda y_{n+1}(1 - ha_{21}) \\ k_3 &= \lambda(y_{n+1} - ha_{31}k_1 - ha_{32}k_2) \\ &= \lambda y_{n+1}(1 - h(a_{31} + a_{32}) + ha_{21}a_{32}) \end{aligned} \tag{3.9}$$

Substituting (3.9) into (3.7) to advance the solution to the next node point, and setting $q = \lambda h$, yields

$$y_{n+1} = y_n + y_{n+1}q[\delta_1 - q\delta_2 + q^2\delta_3] \tag{3.10}$$

where

$$\begin{aligned} \delta_1 &= c1 + c_2 + c_3 \\ \delta_2 &= a_{21} + a_{31} + a_{32} \\ \delta_3 &= a_{21}xa_{32} \end{aligned} \tag{3.11}$$

Hence,

$$\frac{y_{n+1}}{y_n} = \frac{1}{1 - q\delta_1 + q^2\delta_2 - q^3\delta_3} \tag{3.12}$$

Lambert [1973] gives sufficient conditions for a 3-stage ERK method to be 3rd order. Similiarly the following order equations must be satisfied,

$$\begin{aligned} \delta_1 &= 1 \\ \delta_2 &= 1/2 \\ \delta_3 &= 1/6 \end{aligned} \tag{3.13}$$

for the corresponding BRK method to be 3rd order. Thus for the 3-stage

3rd order BRK method

$$E(q) = \frac{y_{n+1}}{y_n} = \frac{1}{1 - q + q^2/2 - q^3/6} \qquad (3.14)$$

Performing this analysis with s-stages will clearly lead to the general form of the stability function,

$$E(q) = \frac{1}{1 + \sum_{j=1}^{s}(-q)^j \delta_j} \qquad (3.15)$$

It will be shown in section 3.4 that this approximation will damp out the fast transients of a component much quicker than the commonly used BDF methods.

Let the stability function of the ERK method be $E_e(q)$. To find the stability region of the corresponding BRK method, it is applied with step -h to predict y_{n+1} from y_n. It follows that the BRK method is merely a case of interchanging y_{n+1} and y_n and replacing h by -h, ie. q by -q. Thus (2.4) becomes

$$\frac{y_n}{y_{n+1}} = E_e(-q) \qquad (3.16)$$

ie

$$\frac{y_{n+1}}{y_n} = \frac{1}{E_e(-q)} = E_b(q) \qquad (3.17)$$

The stability function of the BRK method, $E_b(q)$, is thus expressed in terms of the stability function of the corresponding explicit method.

The stability region of the BRK methods like all Runge-Kutta methods are symmetric about the real axis. Let the general stability function be of the form

$$\frac{N(q)}{D(q)} = \frac{\sum_{j=1}^{t} \tau_j q^j}{\sum_{j=1}^{s} \delta_j (-q)^j} \qquad (3.18)$$

where $N(q) \equiv 1$ for a BRK method and $D(q) \equiv 1$ for an ERK method. To show that the resulting stability region is symmetrical about the real axis substitute $Rexp(i\theta)$ for q in (3.18),

$$\frac{N(Rexp(i\theta))}{D(Rexp(i\theta))} = \frac{\sum_{j=1}^{t} \tau_j R^j \times \{\cos(j\theta) + i\sin(j\theta)\}}{\sum_{j=1}^{s} \delta_j R^j \times \{\cos(j\theta) + i\sin(j\theta)\}}$$

$$= \frac{\sum_{j=1}^{t} \tau_j R^j \cos(j\theta) + i\sum_{j=1}^{t} \tau_j R^j \sin(j\theta)}{\sum_{j=1}^{s} \delta_j R^j \cos(j\theta) + i\sum_{j=1}^{s} \delta_j R^j \sin(j\theta)} \qquad (3.19)$$

Taking the modulus of both sides

$$\frac{|N(Rexp(i\theta))|}{|D(Rexp(i\theta))|} = \frac{\{[\sum_{j=1}^{t} \tau_j R^j \cos(j\theta)]^2 + [\sum_{j=1}^{t} \tau_j R^j \sin(j\theta)]^2\}^{1/2}}{\{[\sum_{j=1}^{s} \delta_j R^j \cos(j\theta)]^2 + [\sum_{j=1}^{s} \delta_j R^j \sin(j\theta)]^2\}^{1/2}} \qquad (3.20)$$

It follows that

$$\frac{|N(Rexp(-i\theta))|}{|D(Rexp(-i\theta))|} = \frac{\{[\sum_{j=1}^{t} \tau_j R^j \cos(-j\theta)]^2 + [\sum_{j=1}^{t} \tau_j R^j \sin(-j\theta)]^2\}^{1/2}}{\{[\sum_{j=1}^{s} \delta_j R^j \cos(-j\theta)]^2 + [\sum_{j=1}^{s} \delta_j R^j \sin(-j\theta)]^2\}^{1/2}}$$

$$= \frac{|N(Rexp(i\theta)|}{|D(Rexp(i\theta)|} \qquad (3.21)$$

hence the region of absolute stability of all Runge-Kutta methods are symmetric about the real axis.

The stability region of a BRK method is closely related to the stability region of the corresponding ERK method. Let $-q_b$ be on the

boundary of the stability region of the ERK method, ie.

$$|E_e(-q_b)| = 1 \tag{3.22}$$

using (3.15)

$$|E_b(q_b)| = \frac{1}{|E_e(-q_b)|} = 1 \tag{3.23}$$

ie. q_b is on the boundary of the stability region of the BRK method. Similiarly if $-q_i$ and $-q_o$ are, respectively inside and outside the stability region of the ERK method, then

$$\text{and} \quad \begin{aligned} |E_b(q_i)| &> 1 \\ |E_b(q_o)| &< 1 \end{aligned} \tag{3.24}$$

Hence from (3.23) and (3.24) the stability region of the BRK method is the complementary set of the image of the stability region of the ERK reflected in the imaginary axis. This is clearly shown by considering Euler (ERK) and Backward Euler (BRK), Figure 3.1.

As discussed in chapter 2, a region of absolute stability of a method gives only limited insight into its potential performance. But it is useful for an indication of the restriction placed on the step size for solving stiff problems. This can be seen by considering the contour R = 1 of the modulus plots in Figures 3.2 - 3.5. These figures show modulus and argument plots for BRK methods of orders 1 to 4.

As with ERK methods, the stability regions of BRK methods of order 1-4 can only be enlarged by the addition of extra costly stages. For 5th and 6th order methods of 6- and 7-stages respectively, however, they each have 1 free parameter. This free parameter can be adjusted to enhance the stability region of the method. It cannot, however, be used to increase the order of the method, ie making this free parameter equal to 1/6!, for the 5th order method, will not meet the order

requirements for a 6th order method. An s-stage s-1 order BRK method will have a stability function of the form

$$E(q) = \frac{1}{1 + \sum_{j=1}^{s-1}(-q)^j/j! + \delta_s(-q)^s} \qquad (3.25)$$

thus δ_s is the free parameter. A numerical search was conducted to find the value of δ_s for which the A(α)-stability region were largest for s = 6 and 7. A value for δ_s was selected, from a plausible range, and the boundary locus method used to locate the boundary of the stability region. When a negative root of (2.15) was located, ie. a point on the boundary in the left-hand half plane, the angle between the real axis and a line from the origin to this point was calculated. The smallest angle, α, was noted for each value of δ_s. When the complete range had been swept the process was repeated with a reduced range, ie. a subset of the original range. This process continued until the value of δ_s to maximize α was determined to a reasonable accuracy. The δ_s values obtained in this manner were,

δ_6 = 2.31e-4 giving α = 79.1° and

δ_7 = -2.31e-5 giving α = 74.3°.

Figures 3.6 and 3.7 show the stability regions of the 5th and 6th order methods with δ_s set as above. It is apparent from these plots, that although the 5th and 6th order BRK methods cannot be made A-stable, the whole of the imaginary axis, in the case of the 6th order method, is included in the stability region. In fact setting δ_6=1.2e-3 allows the whole of the imaginary axis to be included in the stability region of the 5th order method, however, the method is only A(77.2°)-stable. The small regions of instability in the left-hand half plane do not necessarily invalidate the method for solving stiff systems.

Using the 6-stage 5th order method of Lawson[1967], the coefficients of

the method which possesses this optimal stability region can be generated. The coefficients, given in Table 3.1, are in their ERK form, but can be expressed in the fully implicit form by applying the simple transformation (3.5).

3.4 Other stability properties of BRK methods

One other important stability property needed by an integrator of stiff systems is L-stability. The current analysis of L-stability, however, does not indicate to what extent the numerical ratio E(q) damps out the fast transients compared to the analytical solution (2.3), ie. how well the numerical quantity E(q) approximates exp(q). To reproduce the fast decay of exp(q) the numerical ratio, E(q), should have a stability function of the form,

$$\frac{1}{1 + \sum_{j=1}^{s} (-q)^j \delta_j} \qquad (3.26)$$

Clearly no explicit method can do this. The modulus and argument plots of chapter 2, in part, show the damping ability of the method, but they do not explicity define the amount of damping produced. This can be rectified if the L^r-stability of a method is considered, Richards and Everett[1983].

A k-step method is said to be L^r-stable, if when applied to the standard test problem (2.1)

$$\left[\frac{y_{n+1}}{y_n}\right]^{1/k} = O(\ Re(q)^{-r}\ ) \text{ as } Re(q) \to -\infty \qquad (3.27)$$

Clearly any 1-step L-stable method is at least L^1-stable.

The most commonly used methods for solving stiff systems are the BDF methods. The following theorem demonstrates that as the order of these

methods is increased the order of damping produced decreases.

Theorem 3.1 : An order p BDF is $L^{1/p}$-stable.

A brief sketch of the proof is now given : Consider the general pth order BDF applied to (2.1)

$$(1-q\beta_p)y_{n+p} + \sum_{j=0}^{p-1}\alpha_j y_{n+j} = 0 \qquad (3.28)$$

Then the auxiliary equation corresponding to the linear difference equation (3.29) is

$$(1-q\beta_p)r^p + \alpha_{p-1}r^{p-1} + \ldots + \alpha_0 = 0 \qquad (3.29)$$

Let the roots of (3.29) be $\zeta_1, \zeta_2, \ldots, \zeta_p$ which generally will be distinct so that the general solution to (3.28) will be

$$y_n = \sum_{i=1}^{p} A_i \zeta_i^n \qquad (3.30)$$

where A_i are constant. We note that as $Re(q) \to -\infty$, $\zeta_i \to 0$ for all i=1(1)p. It follows that

$$\frac{y_{n+p}}{y_n} = \frac{\sum_{i=1}^{p} A_i \zeta_i^{n+p}}{\sum_{i=1}^{p} A_i \zeta_i^n} = \frac{\sum_{i=1}^{p}\left[A_i \zeta_i^n \sum_{j=0}^{p-1}\alpha_j \zeta_i^j\right]}{(q\beta_p-1)\sum_{i=1}^{p} A_i \zeta_i^n} \qquad (3.31)$$

Hence taking limits of (3.31) as $Re(q) \to -\infty$

$$\text{Limit} \left|\frac{y_{n+p}}{y_n}\right|^{1/p} =$$

$$\text{Limit} \left|\frac{1}{q\beta_p-1}\right|^{1/p} \left| \alpha_0 + \alpha_1 \text{Limit} \frac{\sum_{i=1}^{p} A_i \zeta_i^{n+1}}{\sum_{i=1}^{p} A_i \zeta_i^n} + \ldots \right.$$

$$\left. + \alpha_{p-1}\text{Limit} \frac{\sum_{i=1}^{p} A_i \zeta_i^{n+p-1}}{\sum_{i=1}^{p} A_i \zeta_i^n} \right|^{1/p} \qquad (3.32)$$

By dividing (3.29) by $1-q\beta_p$ and allowing $q \to -\infty$ we see that $\zeta_i \to 0$ for

i=1(1)p. Hence by repeated use of L'Hopitals rule

$$\lim_{\mathrm{Re}(q)\to-\infty}\left|\frac{y_{n+p}}{y_n}\right|^{1/p} = \lim_{\mathrm{Re}(q)\to-\infty}\left|\frac{\alpha_o}{q\beta_p-1}\right| = O\left(\mathrm{Re}(q)^{-1/p}\right) \quad (3.33)$$

Thus a BDF method is $L^{1/p}$-stable. BRK methods on the other hand achieve (3.23) for orders 1 to 4 and hence are L^p-stable. For p greater than four BRK methods give an approximation to exp(q) of the form

$$\frac{1}{1 + \sum_{i=1}^{4}(-q)^i/i! + \sum_{i=5}^{s}(-q)^i\delta_i} \quad (3.34)$$

Hence a general s-stage BRK method is normally L^s-stable but exceptions can be found, ie. if $\delta_s = 0$.

The superior damping properties of BRK methods over the commonly used BDF methods can be demonstrated numerically by applying them both to the standard test problem, (2.1) with λ real and negative and the initial condition y(0) = 1.

The extra starting values required by the BDF (for orders greater than one) are supplied by the analytical solution, (2.2). Two tests were performed, the results of which are displayed in Table 3.2, one with q = -1 and the other introducing more damping by setting q = -10, the analytical solution is also shown for comparison.

For order 1 both BDF and BRK are the same method, Backward Euler and hence the results are the same. When q = -1 a slight improvement in the BRK methods solution is obtained by the use of a higher order, whereas there is no significant improvement in the corresponding BDF solution. Increasing q to -10, however, shows a dramatic improvement in the BRK solution as order is increased compared with the

deterioration of the BDF solution.

3.5 Implementation details

The implementation for a BRK method will now be discussed with particular reference, for simplicity, to a 2-stage 2nd order BRK method. The implementation details apply equally well to other orders.

As BRK methods are implicit, in y_{n+1} only, algebraic equations of the form

$$\epsilon(x_{n+1}, y_{n+1}) \equiv y_{n+1} - y_n - h\sum_{i=1}^{s} c_i k_i = 0 \qquad (3.35)$$

must be solved at each step. There is no guarantee that this equation has a unique solution for y_{n+1}. However, we attempt to find a solution using an iterative process. Using a simple functional iteration process, however, places an unacceptable restriction on the step size. A more robust method must therefore be employed. The method usually employed is based on the Newton iteration process which will usually converge providing that a good initial estimate can be supplied, without severe restrictions on the step size. This leads to the following iterative scheme

$$\frac{\partial \epsilon}{\partial y_{n+1}} \left[y_{n+1}^{(i+1)} - y_{n+1}^{(i)} \right] = -\epsilon(y_{n+1}^{(i)}) \qquad (3.36)$$

where the iteration matrix, $\partial\epsilon/\partial y_{n+1}$, is evaluated at every iteration for a full Newton method and from (3.35) has the form,

$$\frac{\partial \epsilon}{\partial y_{n+1}} = I - h \sum_{i=1}^{s} c_i \frac{\partial k_i}{\partial y_{n+1}}$$

$$\frac{\partial k_1}{\partial y_{n+1}} = \frac{\partial f(y_{n+1})}{\partial y_{n+1}} \qquad (3.37)$$

$$\frac{\partial k_2}{\partial y_{n+1}} = \left[I - h a_{21} \frac{\partial k_1}{\partial y_{n+1}} \right] \frac{\partial f(y_{n+1} - h a_{21} k_1)}{\partial y_{n+1}} \text{ , etc.}$$

where I is the N×N unit matrix. If the ODE under consideration is linear then the Jacobian matrix is constant ie.

$$\frac{\partial f}{\partial y_{n+1}} = J \qquad (3.38)$$

If, however, the problem is non-linear but the solution is slowly varying then the problem can be considered as static in an interval and thus (3.39) still holds. For a 2-stage 2nd order BRK method applied to a linear problem

$$\begin{aligned}\epsilon(y_{n+1}) &= y_{n+1} - y_n + h[c_1k_1 + c_2k_2] \\ &= y_{n+1} - y_n + h[c_1Jy_{n+1} + c_2J(y_{n+1} - hb_{21}Jy_{n+1})] \\ &= [I - hJ + (hJ)^2/2]y_{n+1} - y_n \qquad (3.39)\end{aligned}$$

and the iteration matrix, from (3.37), is

$$(I - hJ + (hJ)^2/2) \qquad (3.40)$$

If the exact iteration matrix, (3.40), is used with exact arithmetic then, using (3.39) and (3.40), the Newton process, (3.36), can be rewritten as,

$$\begin{aligned}y_{n+1} &= y_{n+1} - [I - hJ + (hJ)^2/2]^{-1}\times \\ &\qquad [(I - hJ + (hJ)^2/2)y_{n+1} - y_n] \\ &= [I - hJ + (hJ)^2/2]^{-1}y_n \qquad (3.41)\end{aligned}$$

Thus the process must converge in one iteration. However, the exact iteration matrix is usually unavailable, so an approximation to (3.40) is used. Let this be M, the exact form of M is given later in this section. Using this the Newton process (3.36) becomes

$$\begin{aligned}y_{n+1} &= y_n - [M]^{-1}\times[(I - hJ + (hJ)^2/2)y_{n+1} - y_n] \\ &= [I - [M]^{-1}\times[(I - hJ + (hJ)^2/2)y_{n+1}] - [M]^{-1}y_n \qquad (3.42)\end{aligned}$$

Hence if (3.40) is not exact then an iterative scheme is required. This requires the matrix M to be updated at each iteration, in practice a modified Newton process is used whereby the iteration matrix is only updated when it is strictly necessary. More precise details about the updating of this matrix will be given later in this section.

The actual procedure used to solve the system of equations is obtained by rewriting (3.36), using M as an approximation to $\partial\epsilon/\partial y_{n+1}$, as

$$M\Delta_{i+1} = -\epsilon(y_{n+1}) \qquad (3.43)$$

where

$$\Delta_{i+1} = y_{n+1}^{(i+1)} - y_{n+1}^{(i)} \qquad (3.44)$$

Thus M must be computed as a good approximation to (3.40), especially as it is only updated when necessary. The simplest approximation is to compute (3.40) directly ie. evaluate J^2 and J numerically, by finite differences, and then construct (3.40). However, this has a variety of limitations. The squaring of J, for 2nd order, is acceptable but for higher order methods the evaluation of J^S, J^{S-1}, . . . is not. The storage requirements are also very high even if nested multiplication is used.

To alleviate these problems, the present work involves the direct evaluation of $\partial\epsilon/\partial y_{n+1}$ by numerical differentiation. Each component of y_{n+1} is perturbed by an amount β_i, where the subscript denotes the ith component of the vector y_{n+1}, and the corresponding new value of $\epsilon(y_{n+1})$ computed. Various fixed perturbations were examined eg. β_i = 1.e-10, 1.e-12, but each was found to be unsatisfactory. Hence, some change relative to the ith component was required. To cope with the case of any component of y_{n+1} being zero an absolute perturbation was also added. Thus the perturbation is constructed as a combination of a relative part and an absolute part,

$$\beta_i = y_{n+1}[[1,\sqrt{eps}]] + SIGN(y_{n+1},1)\sqrt{eps} \qquad (3.45)$$

where eps is the smallest positive number representable on the computer for which (1 + eps) is greater than one. The function SIGN(a,b), which returns the sign of a with the magnitude of b, is used to ensure that a zero perturbation is avoided, ie. the relative and absolute changes

always have the same sign.

The solution of (3.43) is then performed by decomposing the iteration matrix into a LU product, L being a lower triangular matrix with unit diagonal and U an upper triangular matrix. This reduces equation (3.43) to

$$LU\Delta_{i+1} = -\epsilon(y_{n+1}) \tag{3.46}$$

which is solved by first solving

$$Uz_{i+1} = -\epsilon(y_{n+1}) \tag{3.47}$$

and then

$$L\Delta_{i+1} = z_{i+1} \tag{3.48}$$

by forward and backward substitutions in the normal manner to yield the displacement vector Δ_{i+1}. Equation (3.44) is then used to generate the next iterative value of y_{n+1}. This process, (3.46) is continued until convergence of Δ_{i+1} is attained.

Convergence can only be attained if successive iterations tend to some limit, which ultimately depends upon the approximation M. Consider two successive iterations

$$y_{n+1}^{i+1} = My_{n+1}^{i} + c \tag{3.49}$$

$$y_{n+1}^{i} = My_{n+1}^{i-1} + c \tag{3.50}$$

using the displacement vector, (3.44), and subtracting (3.50) from (3.49)

$$\begin{aligned}\Delta_i &= M\Delta_{i-1} = M(M\Delta_{i-2}) = \ldots \\ &= M^i\Delta_o\end{aligned} \tag{3.51}$$

Assuming that there exists an exact solution y_{n+1} and defining a residual vector as

$$r_i = y_{n+1}^{i} - y_{n+1} \tag{3.52}$$

then

$$r_i = Mr_{i-1} = M^i r_o \qquad (3.53)$$

Now r_o and Δ_o can be expressed as a linear combination of the eigenvectors of J, μ_j, assuming that they form a basis. Let

$$r_o = \sum_{j=1}^{N} \alpha_j \mu_j \quad \text{and} \quad \Delta_o = \sum_{j=1}^{N} \beta_j \mu_j \qquad (3.54)$$

Clearly generating successive iterations brings in the eigenvalues of M, thus

$$r_i = \sum_{j=1}^{N} \alpha_j \lambda_j^i \mu_j \quad \text{and} \quad \Delta_i = \sum_{j=1}^{N} \beta_j \lambda_j^i \mu_j \qquad (3.55)$$

So clearly for r_i and Δ_i to converge λ_j must be less than 1 for all j=1(1)N, ie. the spectral radius of M, $\rho(M)$, must be less than 1. This is also referred to as the amplification factor, Cash and Singhal[1983].

To examine whether two successive iterations have converged to an acceptable amount, as Δ_{i+1} is unlikely to equal zero, a convergence test must be applied. As the Newton process must be solved accurately the error control must be kept very tight, using a relative error test will ensure that this happens. Thus convergence of the Newton process is considered achieved when

$$\frac{\|\Delta_{i+1}\|}{\text{Max}[\sqrt{eps}, \|y_{n+1}\|]} \leqslant \text{stol} \qquad (3.56)$$

where stol is some acceptable local convergence level.

This check, however, causes one more iteration of the Newton process than is really necessary for the convergence test to be met. Convergence is attained when (3.56) is satisfied, ie. $\|\Delta_{i+1}\|$ must be relatively small, compared with $\|y_{n+1}\|$, hence the addition of this vector to y_{n+1} will make no significant difference to the solution.

Therefore it is advantageous to estimate $\|\Delta_{i+1}\|$ on the ith iteration. Consider

$$y_{n+1}^{i+1} = My_{n+1}^{i} + c \tag{3.57}$$

Using the result of (3.55) and assuming that $|\lambda_1| > |\lambda_j|$ j=2(1)N ie. λ_1 is dominant, then on the jth iteration

$$\Delta_j \cong \beta_1 \lambda_1^j \mu_1 \tag{3.58}$$

and hence

$$\Delta_{j+1} \cong \beta_1 \lambda_1^{j+1} \mu_1 = \lambda_1 (\beta_1 \lambda_1^j \mu_1)$$

$$= \lambda_1 \Delta_j \tag{3.59}$$

Taking a suitable norm, results in

$$\frac{\|\Delta_{j+1}\|}{\|\Delta_j\|} \cong |\lambda_1| = \rho(M) \tag{3.60}$$

Hence the convergence rate of the process is approximately the ratio of two successive displacement vectors which in turn is approximately the spectral radius of the iteration matrix. Let the convergence rate of the process on the ith iteration be $CRATE_i$, thus

$$\frac{\|\Delta_{i+1}\|}{\|\Delta_i\|} = CRATE_{i+1} \tag{3.61}$$

and then

$$\|\Delta_{i+1}\| = \|\Delta_i\| \times CRATE_{i+1} \tag{3.62}$$

Hence the value of $\|\Delta_{i+1}\|$ can be obtained on the ith iteration if an estimate of the convergence rate of the process is obtainable. It can be assumed that the convergence rate on the (i+1)th iteration, $CRATE_{i+1}$, differs from $CRATE_i$ by only a factor of two, ie. convergence is ultimately linear. In any case $CRATE_i$, i = 1, 2, .. must be less that one to ensure convergence. Hence

$$CRATE_{i+1} \leqslant \min(1, 2 \times CRATE_i) \tag{3.63}$$

Thus $\|\Delta_{i+1}\|$ in (3.62) can be approximated by

$$\|\Delta_{i+1}\| = \min(1, 2 \times CRATE_i) \times \|\Delta_i\| \tag{3.64}$$

which often saves at least one iteration of the Newton process. Initially $CRATE_i$ is set to one and updated subsequently by

$$CRATE_i = \max(.2 \times CRATE_{i-1}, \|\Delta_i\| / \|\Delta_{i-1}\|) \qquad (3.65)$$

On the first iteration of each new step the convergence rate from the last iteration of the last step is used. Hence the displacement vector is never calculated on the (i+1)th iteration but always estimated, and it is this estimated value that is always used.

This idea can be extended to check for divergence of the Newton process. If $\|\Delta_{i+1}\| > 10 \times \|\Delta_i\|$, (10 to allow some small increase), then the scheme is showing signs of divergence and some correcting procedure must be forthcoming. This idea was first proposed by Hindmarsh[1974] and later incorporated in most codes that solve systems of equations by a Newton process.

Clearly there will be occasions when the Newton process is going to be slow to converge eg. when too large a step is used. For this reason some limit must be imposed on the maximun number of iterations used. As a failure of the iteration process must lead to a step reduction and almost certainly an expensive iteration matrix update, the scheme is allowed to iterate up to ten times. This figure is, however, rarely reached as the number of iterations usually does not exceed three or four.

The system (3.12) can only be solved for Δ_{i+1} providing the iteration matrix, as it is computed numerically, is non-singular, or more precisely is considered sufficiently well-conditioned for the LU factorization algorithm to be successful. Any single occurrence of it being singular can be overcome by changing the step size. An increase

is not advocated as the local accuracy requirements may not then be met, thus h is reduced. The reduction is performed by halving the step size and recomputing the complete step. If the iteration matrix is repeatedly computed as singular then some other action must be forthcoming, this problem is discussed in detail in chapter 5.

Once the iteration matrix, M , has been evaluated it is the LU factorization which is stored (ignoring the unit diagonal on L) for future use and not M itself. As the iteration matrix is computed numerically, knowledge of the analytical Jacobian matrix J of the initial value problem, would be of no help in the solution procedure.

The iteration matrix is updated for only one of three reasons, viz.

i) the relative change in the step size exceeds 10%,

ii) the iterations fail to converge after three iterations, or

iii) the iteration process shows signs of divergence.

Due to (3.38) being a polynomial in hJ of degree s it is evident that any change in the step size will require a re-evaluation of the matrix M, for good convergent properties to be maintained. This has serious implications in step control policies and they are discussed in chapter 4.

The initial approximaton y^{o}_{n+1} to y_{n+1} is generated by extrapolation using a divided difference table that is constructed from consecutive successful steps of the method. Numerical tests were used to determine the optimal order of the divided difference table for BRK methods of order 2, 5 and 8, these are 3, 7 and 10 respectively.

3.6 Problems considered and numerical results

To confirm the potential shown in BRK methods, fixed step versions of orders 2, 5 and 8 were implemented. As no error estimate was being incorporated the initial and maximum step sizes were supplied as data, to enable a crude error control policy. Step increases by a factor of ten were allowed if the Newton process converged in less than five iterations with a stopping tolerance, stol, of 1.e-10. (The Hindmarsh process described in section 3.5 was not implemented at this stage)

The BRK methods have been applied to a large number of problems and their performance compared to the BDF methods of Gear, incorporated in the NAG library routine D02QBF, Gladwell[1974]. Hereafter GEAR refers to these methods, using a relative error test (CIN(2) = 2), and numerically evaluated Jacobian matrix. The default initial step size was used in all cases. Results are shown for four typical problems, three non-linear and one linear.

In all the following tables and figures the abbreviations below are used,

TOL : local tolerance to satisfy at each step

Steps : number of steps taken to complete integration

FE : total number of function evaluations (including those required for the Jacobian matrix evaluation)

JE : total number of Jacobian evaluations

CPU time : CPU time in seconds on a Prime 550

Order : Order of the method used

Sig. Figs. : significiant figures accuracy computed as

$$-\log_{10} = \text{Min}\left[\frac{|y_N - y_A|}{|y_N|}, 0.\right] \qquad (3.66)$$

where y_N is the numerical solution and y_A the corresponding analytical solution, over the relevant component. The relevant components being y_3, y_3, y_4 and y_1 for problems p3.1, p3.2, p3.3 and p3.4 respectively

The four systems of ODEs considered are listed below together with initial conditions, integration range, eigenvalues of the Jacobian matrix and where appropriate the analytical solution.

$$\frac{dy_1}{dx} = -10^6 y_1 + y_2^2 + y_3^2 - 1 - 1/(1+x)^2$$

$$\frac{dy_2}{dx} = -y_2 + y_3^2(1+x)^2 \qquad x\in[0,10] \qquad \text{(p3.1)}$$

$$\frac{dy_3}{dx} = -y_3^2$$

Initial conditions : $y(0) = [1,1,1]^T$

Eigenvalues : -10^6, -1, $-2/(1+x)$

Analytical solution : $y_1(x) = \exp(-10^6 x)$

$$y_2(x) = 1.0$$

$$y_3(x) = 1/(1+x)$$

This non-linear problem has a stiffness ratio of approximately 1.e6. Initially all components of the solution are of the same magnitude but as the solution procedes y_1 decays rapidly to zero.

$$\frac{dy_1}{dx} = -y_1 y_3 e^x$$

$$\frac{dy_2}{dx} = \frac{-y_2}{(1+x)} \qquad x\in[0,10] \qquad \text{(p3.2)}$$

$$\frac{dy_3}{dx} = -y_2(1+x)e^{-x}$$

Initial conditions : $y(0) = [1e\text{-}2,\ 1e6,\ 1e6]^T$

Eigenvalues : $-10^6,\ -1/(1+x),\ 0$

Analytical solution : $y_1(x) = 10^{-2}\exp(-10^6 x)$

$$y_2(x) = 10^6/(1+x)$$

$$y_3(x) = 10^6\exp(-x)$$

This problem was constructed so that y_1 has an extremely rapid transient but, unlike p3.1 is much smaller in magnitude than the other components. This situation is very common in engineering problems especially where variables are converted to SI units, eg. converting to Pascals may result in some components having extremely large magnitudes, whereas others are extremely small.

$$\frac{dy_1}{dx} = -y_1 + 2$$

$$\frac{dy_2}{dx} = -10y_2 + 20y_1^2$$

$$\frac{dy_3}{dx} = -40y_3 + 80(y_1^2 + y_2^2) \qquad x\in[0,20] \qquad \text{(p3.3)}$$

$$\frac{dy_4}{dx} = -100y_4 + 200(y_1^2 + y_2^2 + y_3^2)$$

Initial conditions : $y(0) = [1,1,1,1]^T$

Eigenvalues : -1, -10, -40, -100

This is a non-linear problem considered by Cash[1975].

$$\frac{dy_1}{dx} = y_2 \qquad x\in[0,10] \qquad \text{(p3.4)}$$

$$\frac{dy_2}{dx} = -10001y_1 - 2y_2$$

Initial conditions : $y(0) = [1,-1]^T$

Eigenvalues : $-1 \pm 100i$

Analytical solution : $y_1(x) = e^{-x}\cos(100x)$

$$y_2(x) = -e^{-x}[\cos(100x) + 100\sin(100x)]$$

This is the only linear problem considered here and is characterised by a highly oscillatory component. It is well known that BDF methods perform particularly badly on such problems.

The initial fast transient stage of a stiff system is usually the most demanding for any stiff integrator, as they only start to work efficiently when the step size is large. The superior damping properties of the BRK methods, over the BDF methods, during this fast transient stage are apparent from Figure 3.8 and Table 3.3 for problem p3.1. In each case the accuracy of the first component is plotted closest to 1.e-6, 2.e-6 and 3.e-6. This ensures that no additional errors are introduced by interpolation. The numbers at the right-hand end of the graph are the number of steps and function evaluations respectively, these clearly show the supremacy of BRK methods at controlling the fast transients. In particular the 8th order method performs extremely well, being able to integrate upto 3.e-6 in just one step and still producing an acceptable solution.

Table 3.5, Figure 3.9, summarises the three BRK methods and GEAR over the whole integration range. The runs were set up so that each took, approximately the same CPU time. All the BRK methods, with the exception of the 2nd order method, out performed the GEAR method on this problem.

Table 3.5 and Figure 3.10 summarises the performance of the methods over the fast transient stage of the solution for problem p3.2. With GEAR it is clear that a very small step must be used initally and maintained for this part of the integration range, whereas the BRK methods can employ a much larger step and still damp out the component. Even with this small step the solution produced by the BDF methods lose accuracy as x increases, this does not happen with the BRK methods.

Integrating over the whole range, Table 3.6 and Figure 3.11, indicates that not only are the BRK methods faster but they are more accurate throughout the range. Due to its poor performance on this 2nd order BRK method is omitted.

Problem p3.3 results are summarised in Table 3.7, Figure 3.12. Even though GEAR is able to used a larger step than any of the BRK methods it is still slower and less accurate, with the exception of the 2nd order BRK method. The somewhat erratic accuracy of all BRK methods can be partially overcome by using a small initial step, but it does highlight the need for some form of error control and variable step policy.

Problem p3.4 is known to severely tax codes based on BDF methods such as GEAR. This is due to the problem being characterized by a highly oscillatory component. Table 3.8, Figure 3.13, show results for, approximately, comparable CPU time. An interesting feature is the high number of Jacobian evaluations required for the BDF code, this suggests that the step control policy is allowing the step to change too freely. Clearly the BRK methods out perform GEAR on this problem.

0.0						
0.5	0.5					
0.25	0.1875	0.0625				
0.5	-0.32392	0.17608	0.14784			
0.75	-0.13206	-0.31965	0.63912	9/16		
1.0	0.444708	0.87328	0.25344	-12/7	8/7	
	7/90	0	32/90	12/90	32/90	70/90

Table 3.1 : Coefficients for 6-stage 5th order ERK.

q	No. of steps	Order	Final values BRK	BDF	Analytical
-1	20	1	9.5367e-7	9.5367e-7	2.0612e-9
		2	1.0995e-8	-4.0863e-6	
		3	3.0243e-9	3.3444e-7	
		4	2.2180e-9	4.6533e-7	
-10	20	1	1.4864e-21	1.4864e-21	1.3839e-87
		2	1.9652e-36	1.5316e-12	
		3	7.1451e-48	3.6652e-10	
		4	6.5734e-57	8.6177e-8	

Table 3.2 : Comparison of BRK and BDF on standard test problem

Method & Order	Log_{10} TOL	Steps at order 1	2	3	4	5	8	FE	JE	Steps	Initial Step
BRK 8	-						30	228	1	30	1.0e-7
GEAR	-15	3	5	17	18	646		869	44	689	7.2e-17
BRK 8	-						12	312	1	12	2.5e-7
GEAR	-12	3	4	15	18	102		383	25	242	1.5e-12
BRK 5	-					60		402	1	60	5.0e-8
BRK 8	-						3	104	1	3	1.0e-6
BRK 5	-					12		162	1	12	2.5e-7
GEAR	-7	3	4	10	12	22		133	11	51	4.9e-10
BRK 8	-						1	65	1	1	3.0e-6
BRK 5	-					2		42	1	2	1.5e-6
BRK 2	-		10					42	1	10	3.0e-7
GEAR	-2	3	4	4				42	4	11	1.2e-8

Table 3.3 : Transient phase of problem p3.1 (Figure 3.8)

Method & order	Log_{10} TOL	FE	JE	Steps	Initial Step	Maximum Step	CPU Time
BRK 2	-	432	1	201	5.0e-2	5.0e-2	1.02
BRK 5	-	1830	6	204	5.0e-2	5.0e-2	2.03
BRK 8	-	2821	2	101	1.0e-1	1.0e-1	3.07
GEAR	-7	424	33	241	4.9e-10	3.6e-1	3.43

Table 3.4 : Full range of problem p3.1 (Figure 3.9)

Method & Order	Log_{10} TOL	Steps at order 1	2	3	4	5	8	FE	JE	Steps	Initial Step
BRK 8	-						20	468	1	20	5.0e-7
BRK 5	-					100		636	1	100	1.0e-7
BRK 8	-						10	299	1	10	1.0e-6
GEAR	-15	2	4	10	14	60		188	15	91	2.4e-10
BRK 5	-					20		204	1	20	5.0e-7
BRK 5	-					10		138	1	10	1.0e-6
BRK 2	-		100					212	1	100	1.0e-7
GEAR	-12	3	4	7	18	3		96	9	35	1.5e-8
BRK 2	-		20					72	1	20	5.0e-7
GEAR	-10	3	7	5				42	5	15	1.5e-7
GEAR	-7	4						19	3	4	1.9e-7
BRK 8	-						2	91	1	2	5.0e-6

Table 3.5 : Transient phase of problem p3.2 (Figure 3.10)

Method & order	Log_{10} TOL	FE	JE	Steps	Initial Step	Maximum Step	CPU Time
BRK 5	-	2166	6	105	1.0e-1	1.0e-1	2.91
BRK 8	-	3783	9	106	1.0e-1	1.0e-1	4.59
GEAR	-10	520	29	385	1.5e-7	6.8e-2	5.16

Table 3.6 : Full range of problem p3.2 (Figure 3.11)

Method & order	Log_{10} TOL	FE	JE	Steps	Initial Step	Maximum Step	CPU Time
BRK 2	-	1160	23	211	1.0e-12	1.0e-1	2.64
BRK 5	-	2640	8	203	1.0e-4	1.0e-1	3.49
BRK 8	-	3263	10	80	2.5e-1	2.5e-1	3.46
GEAR	-6	383	23	206	4.8e-6	1.05e0	4.40

Table 3.7 : Full range of problem p3.3 (Figure 3.12)

Method & order	Log_{10} TOL	FE	JE	Steps	Initial Step	Maximum Step	CPU Time
BRK 2	-	46630	1	20001	5.0e-4	5.0e-4	78.8
BRK 5	-	76044	1	10001	1.0e-3	1.0e-3	63.5
BRK 8	-	130130	1	10001	1.0e-3	1.0e-3	89.9
GEAR	-7	11481	623	9794	4.0e-7	3.1e-3	99.5

Table 3.8 : Full range of problem p3.4 (Figure 3.13)

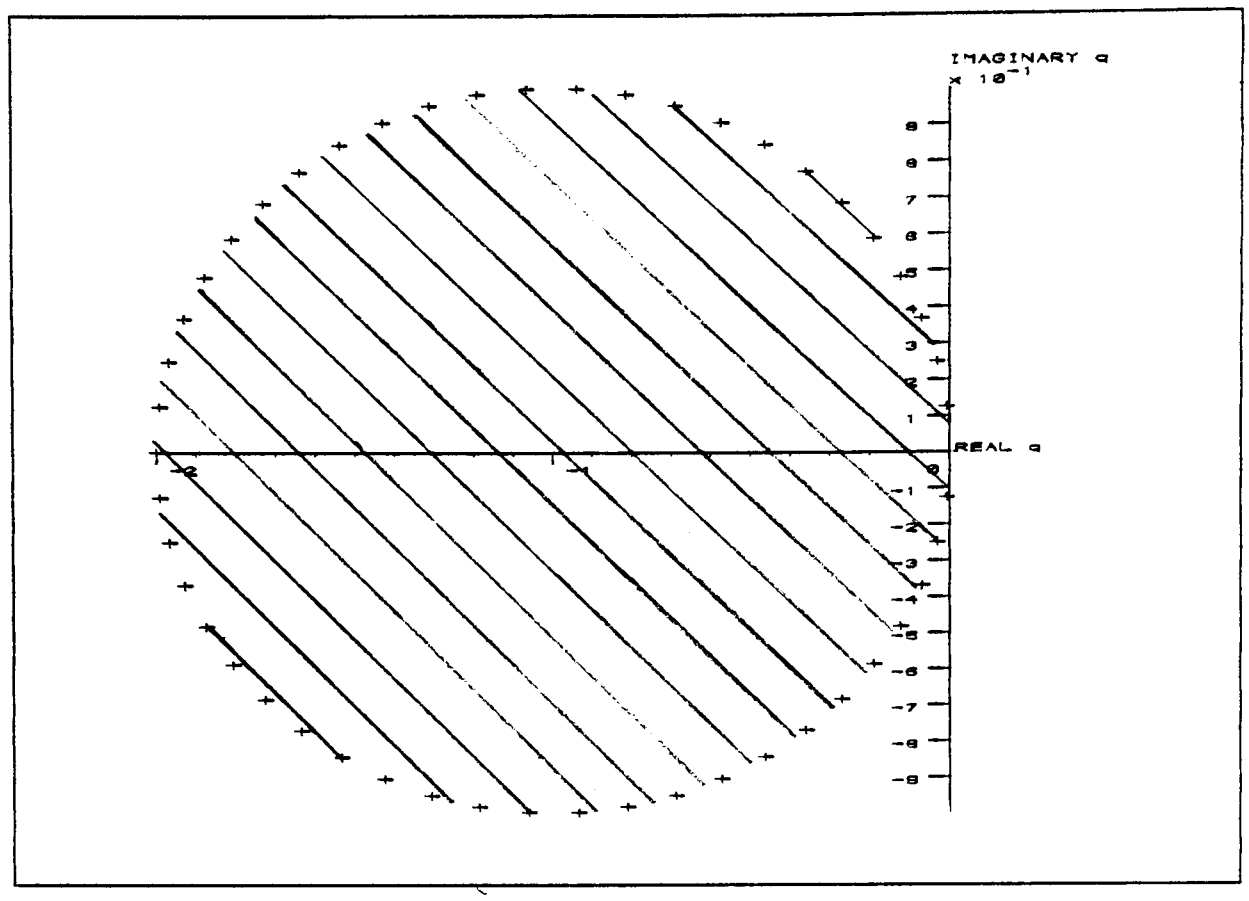


Figure 3.1a : Stability region of Eulers method.

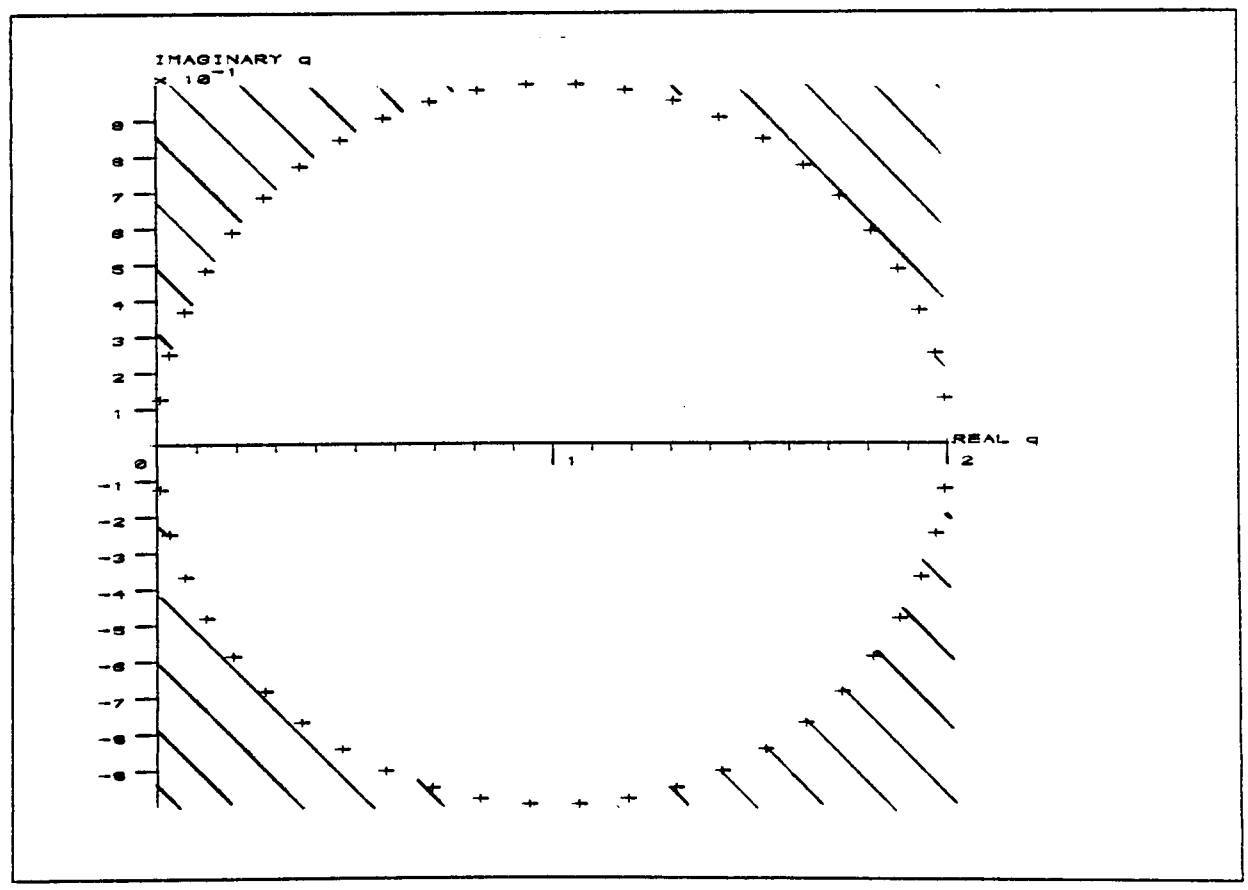


Figure 3.1b : Stability region of Backward Euler method.

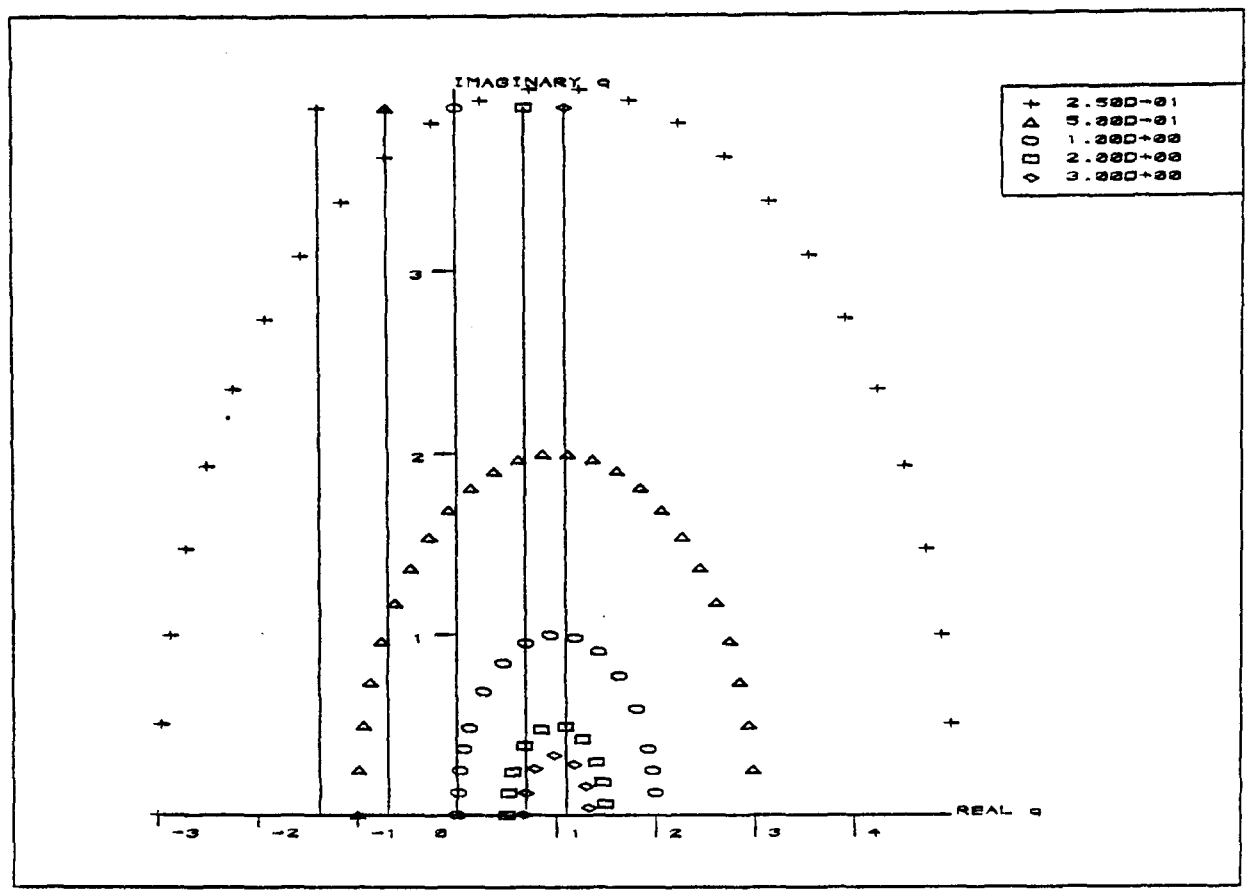


Figure 3.2a : Modulus plot of BRK 1.

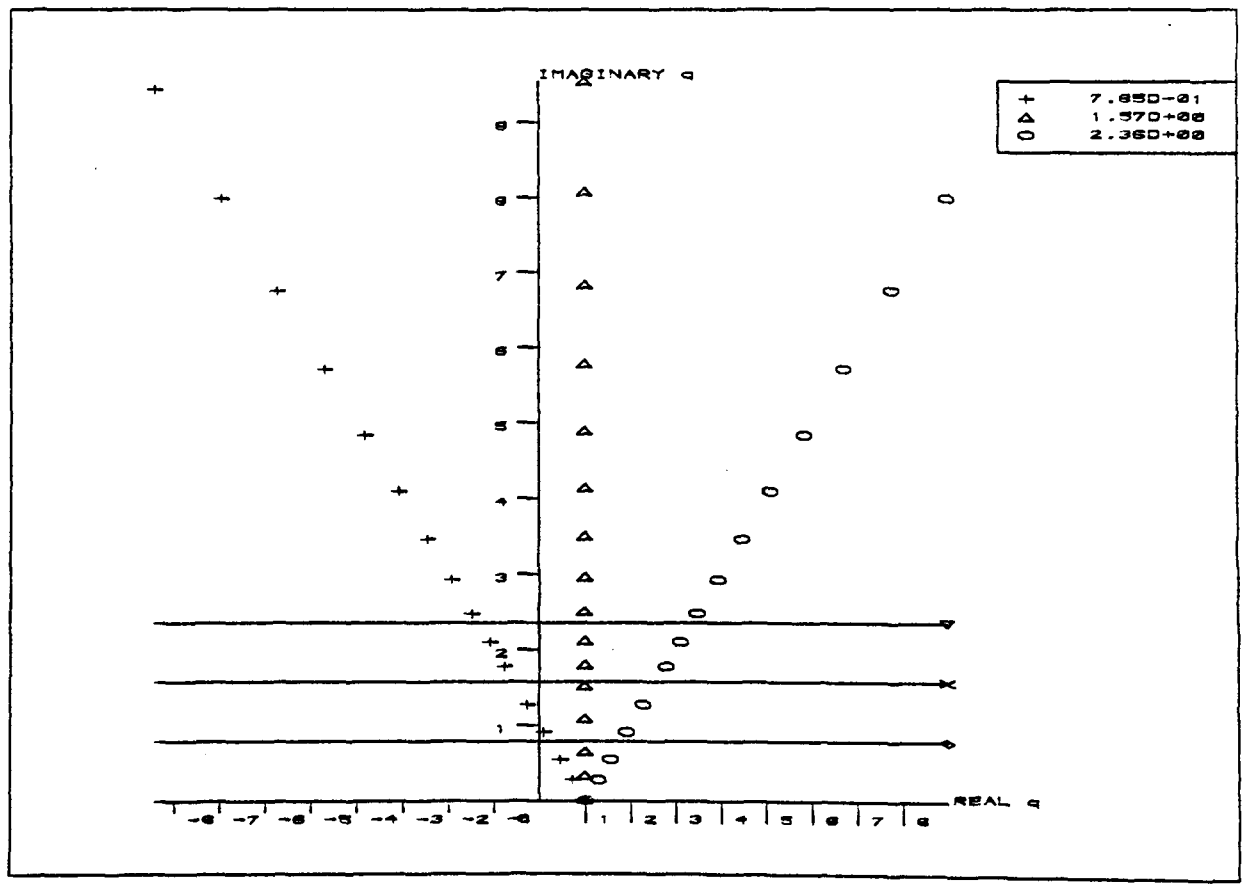


Figure 3.2b : Argument plot of BRK 1.

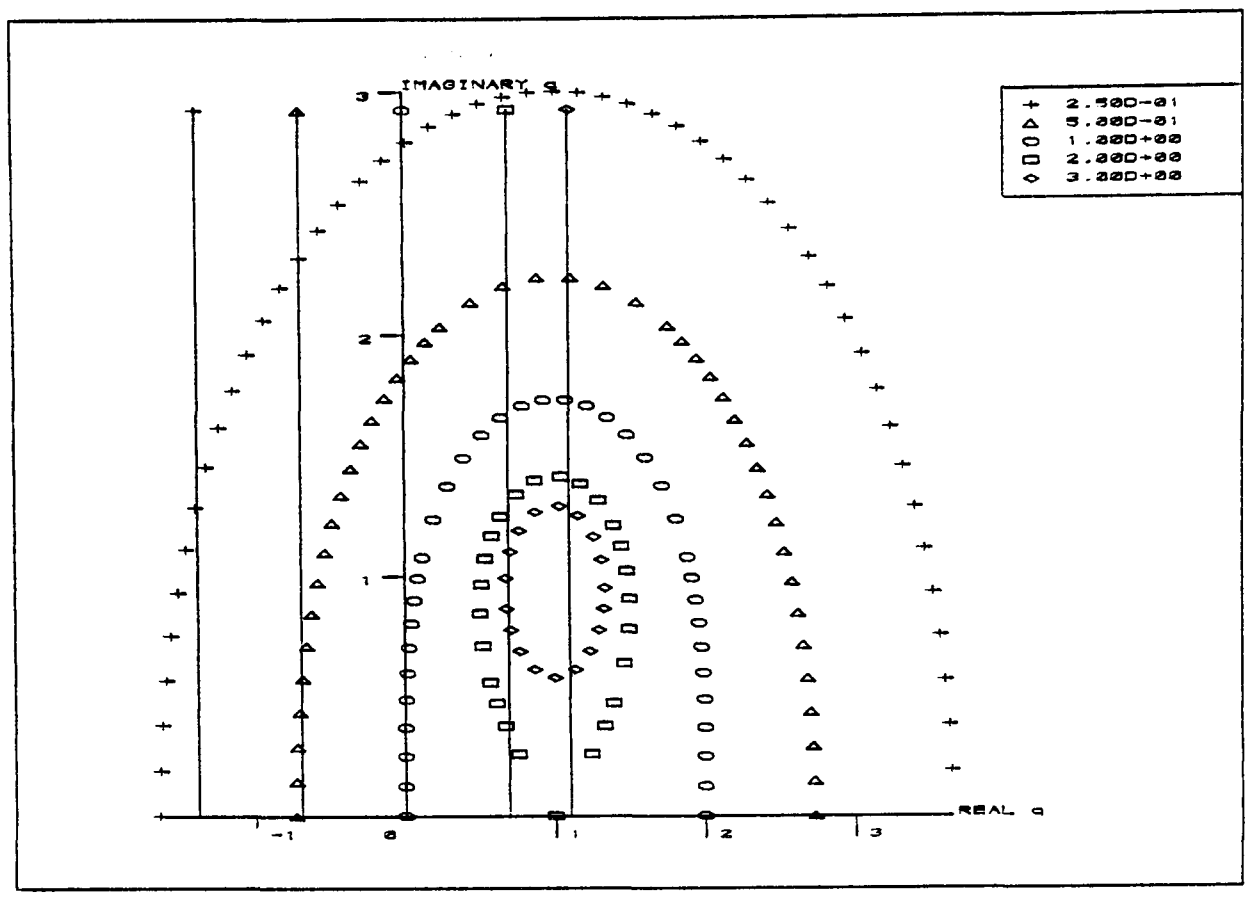


Figure 3.3a : Modulus plot of BRK 2.

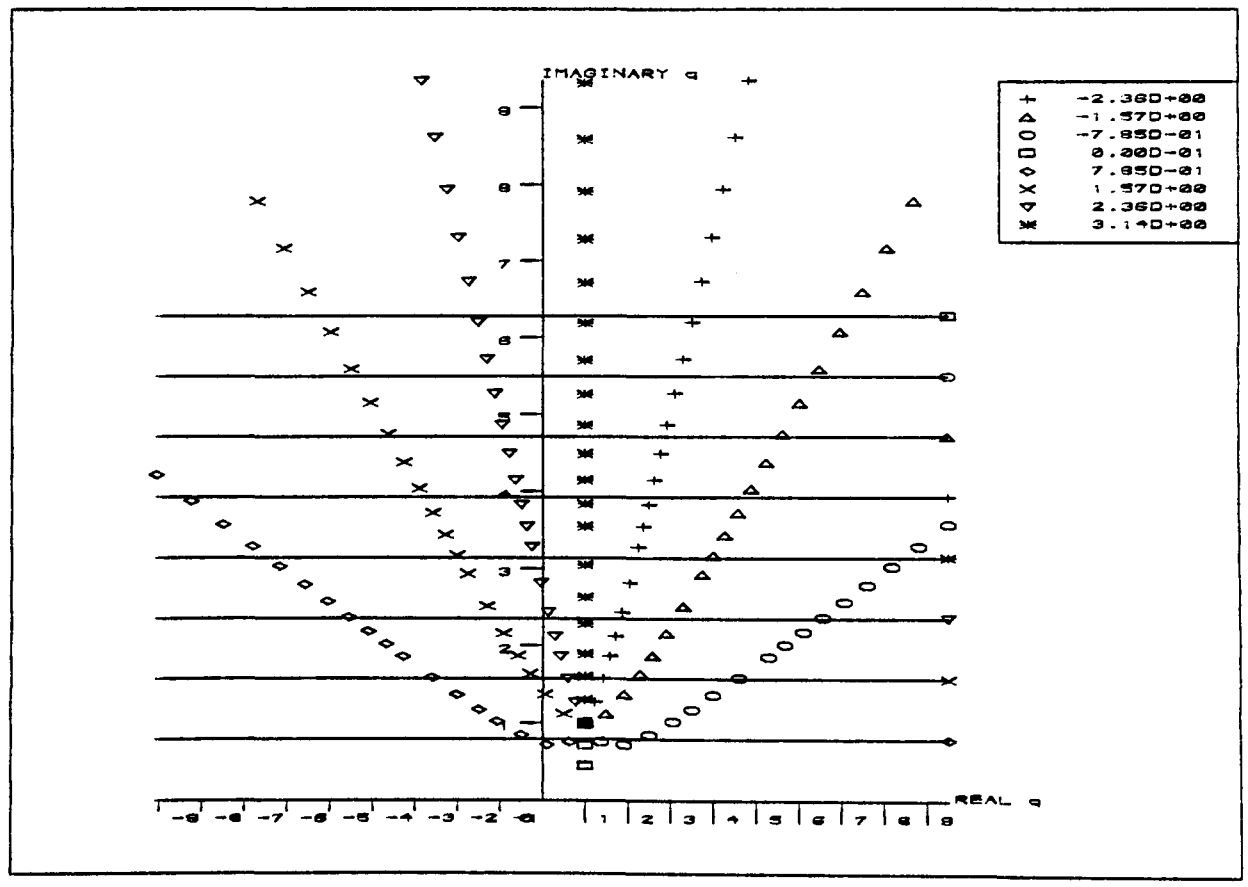


Figure 3.3b : Argument plot of BRK 2.

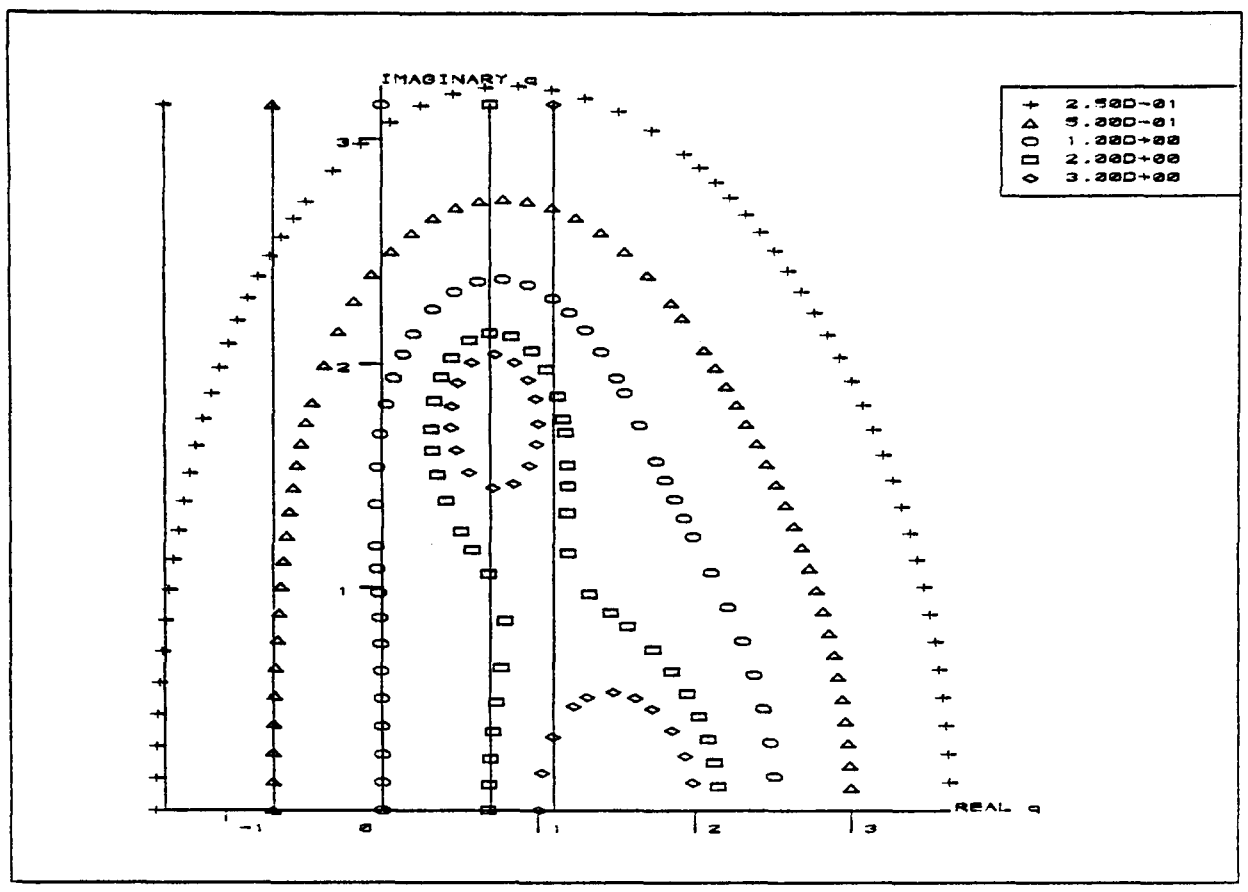


Figure 3.4a : Modulus plot of BRK 3.

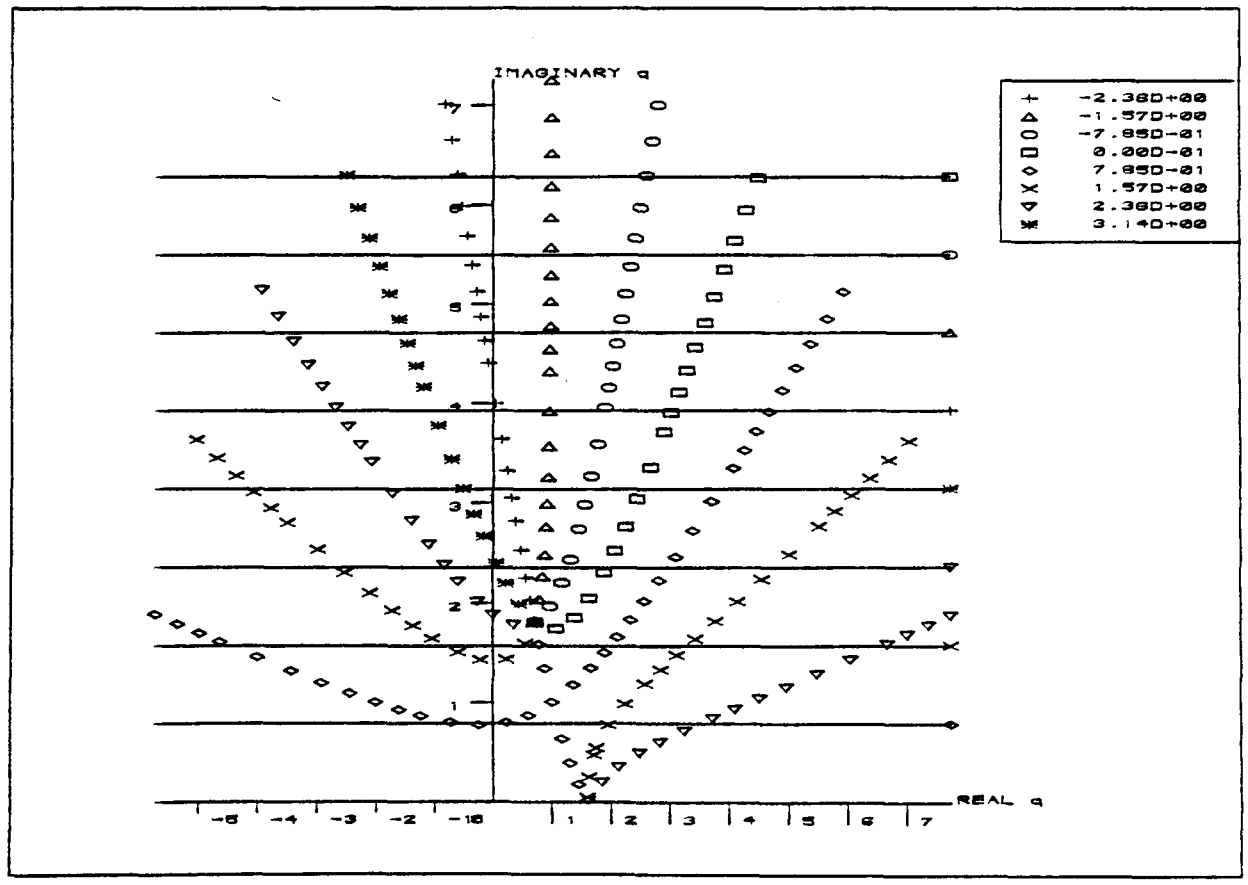


Figure 3.4b : Argument plot of BRK 3.

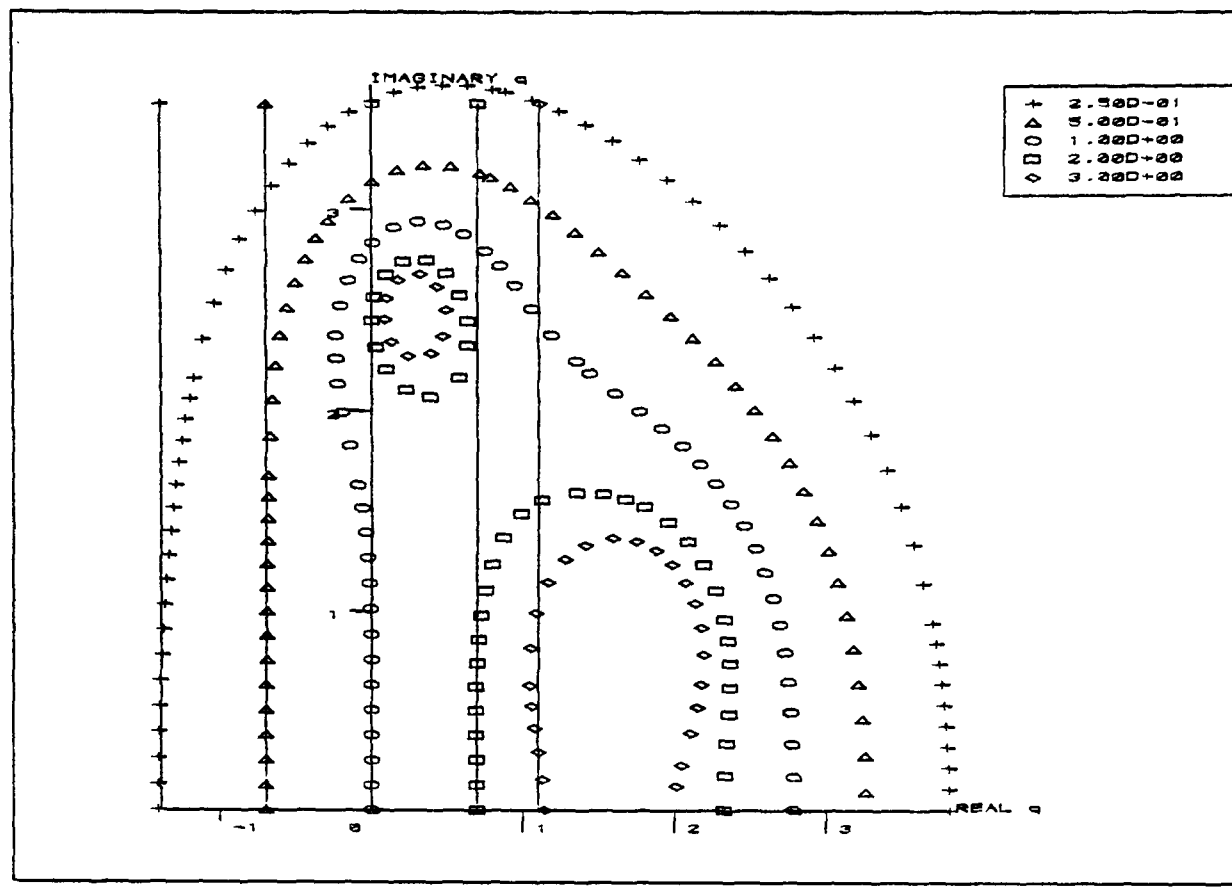


Figure 3.5a : Modulus plot of BRK 4.

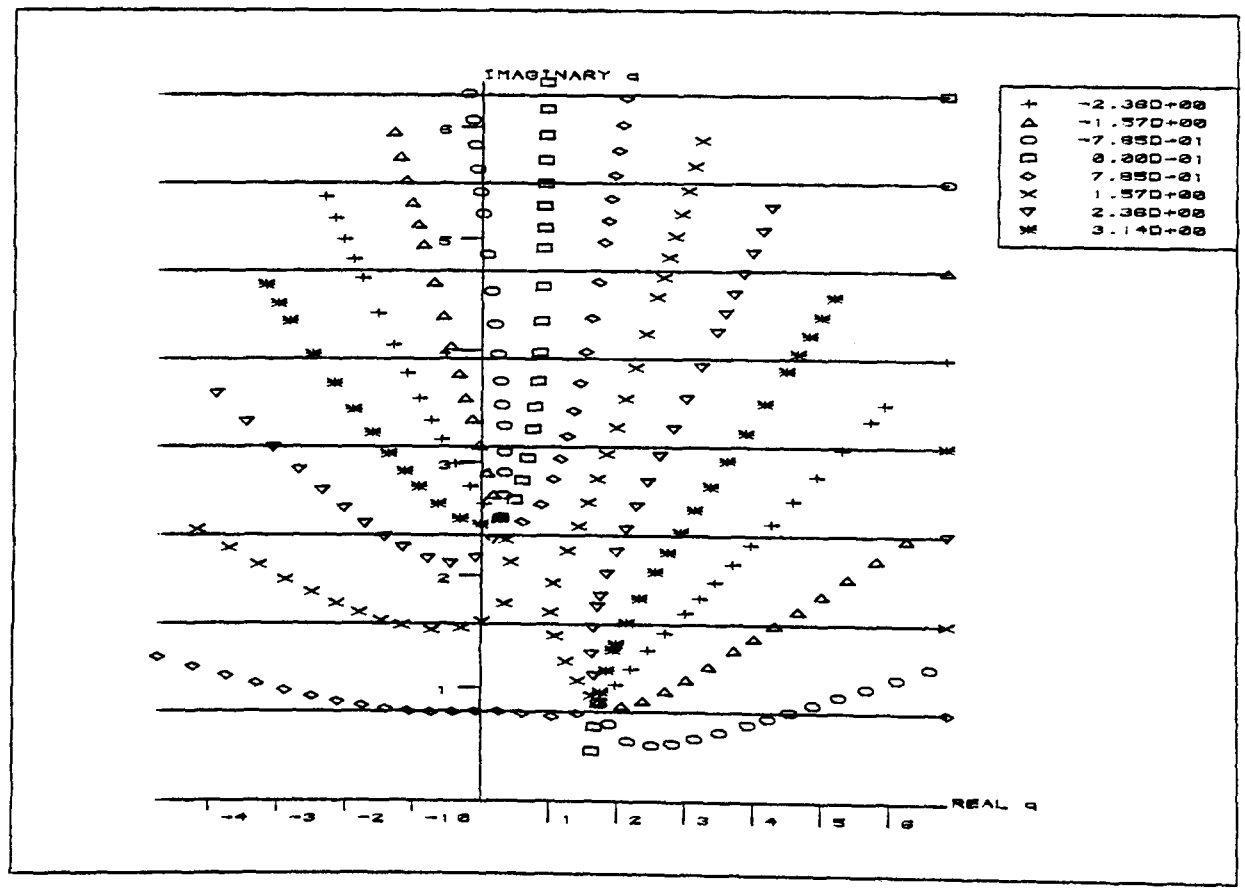


Figure 3.5b : Argument plot of BRK 4.

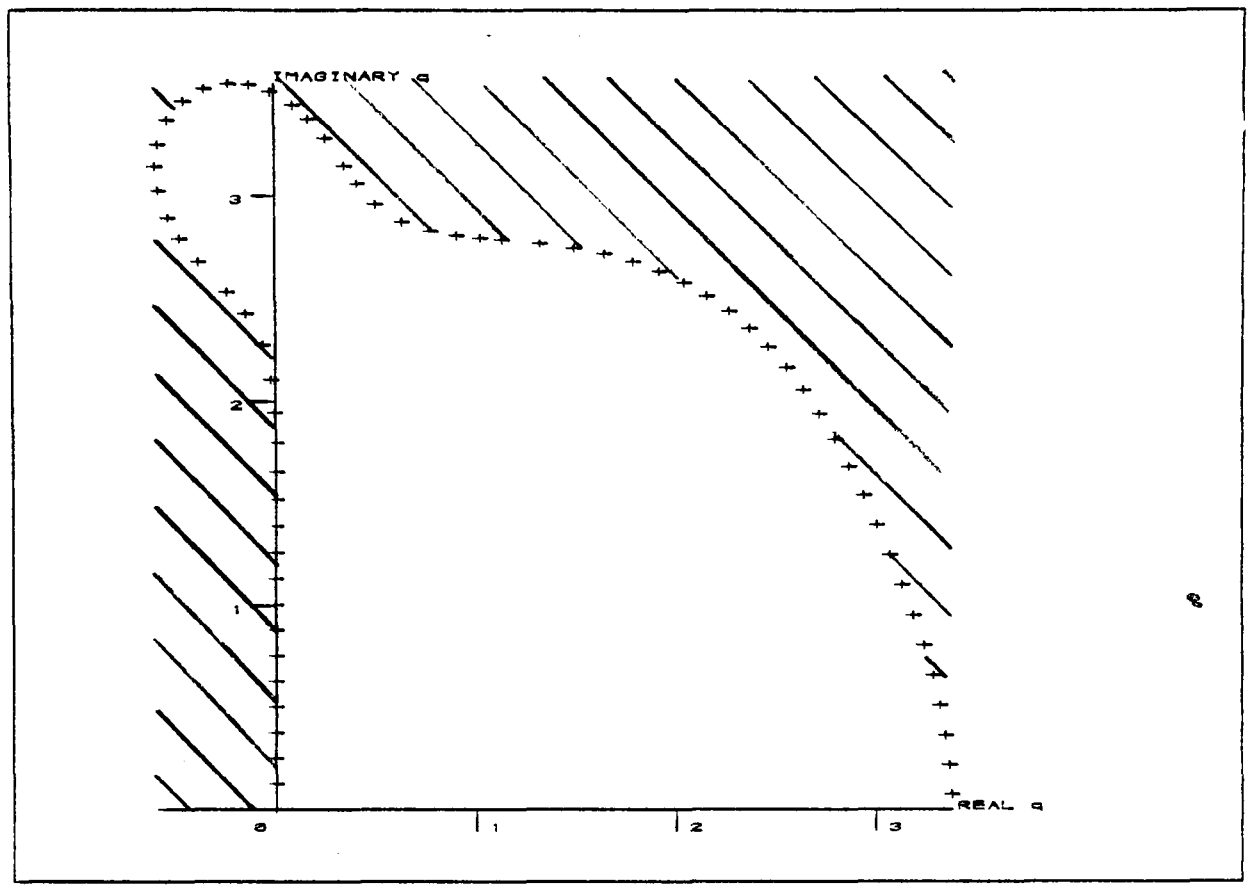


Figure 3.6 : Optimal stability region of 5th order BRK.

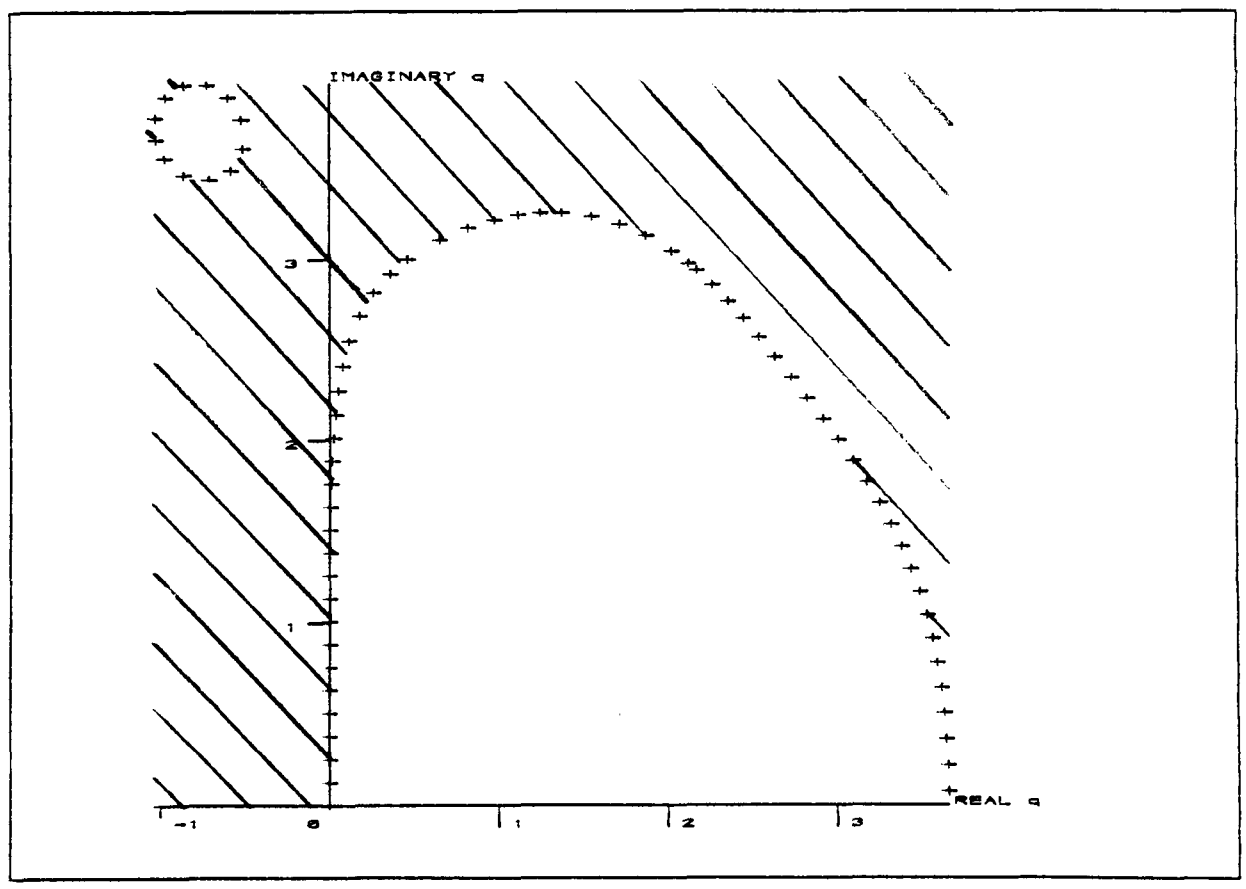


Figure 3.7 : Optimal stability region of 6th order BRK.

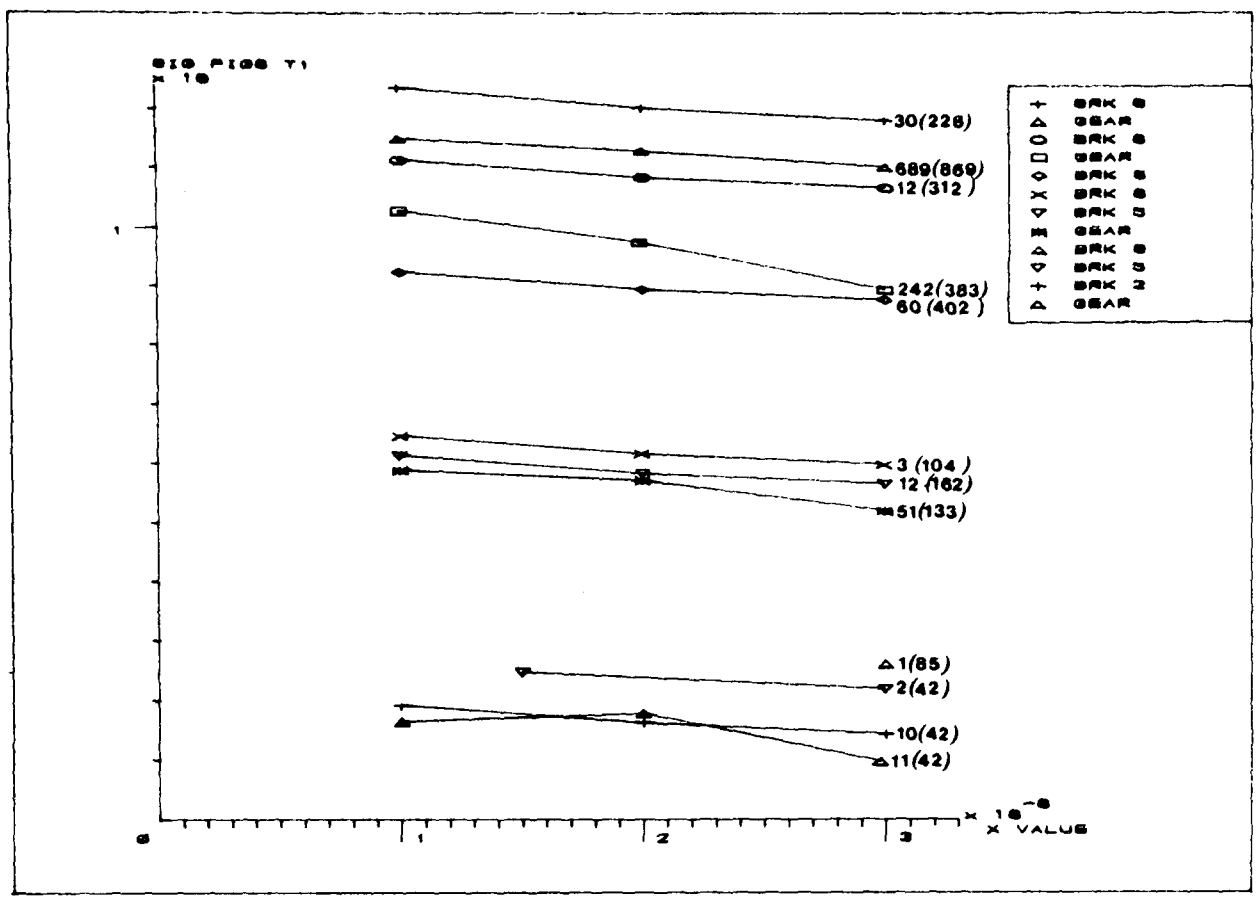


Figure 3.8 : Problem p3.1 over initial stage.

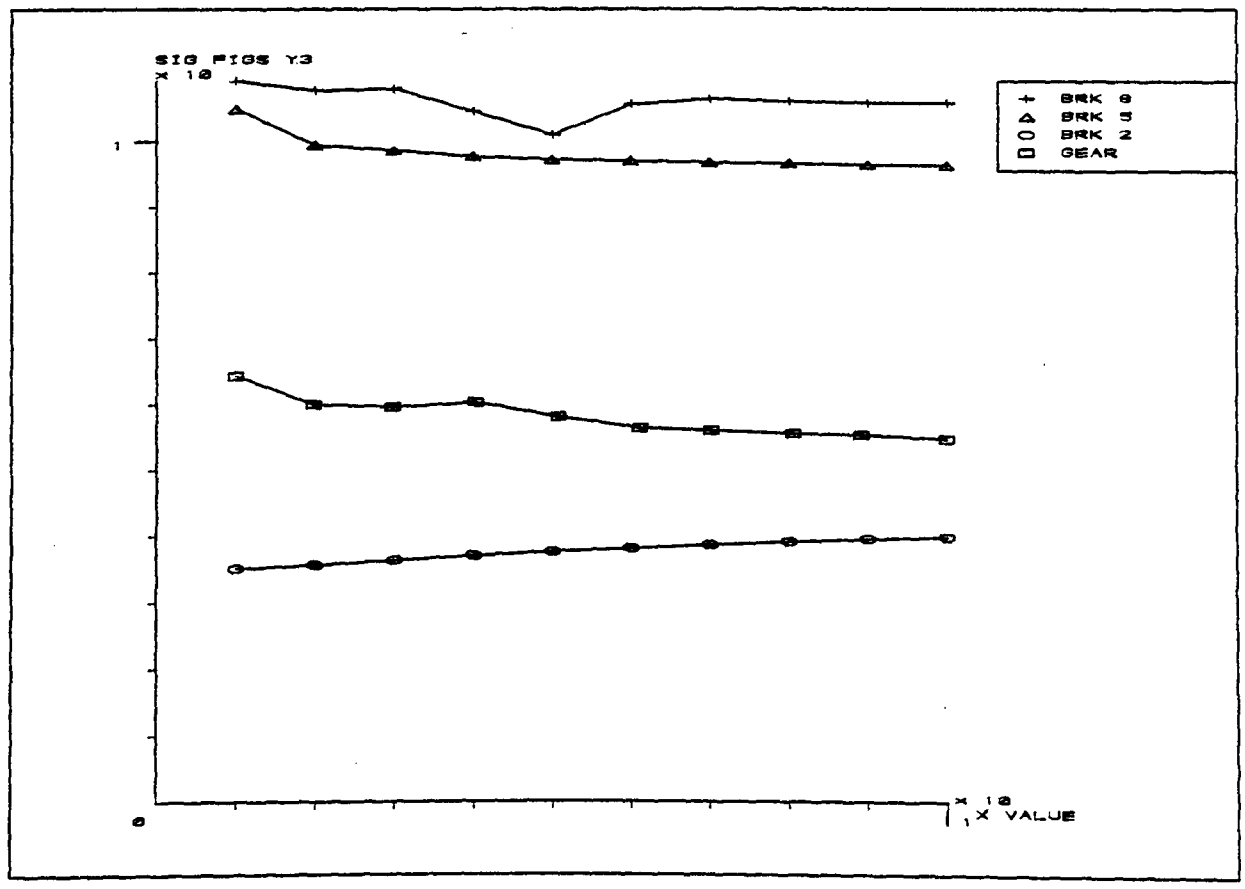


Figure 3.9 : Problem p3.1 over whole range.

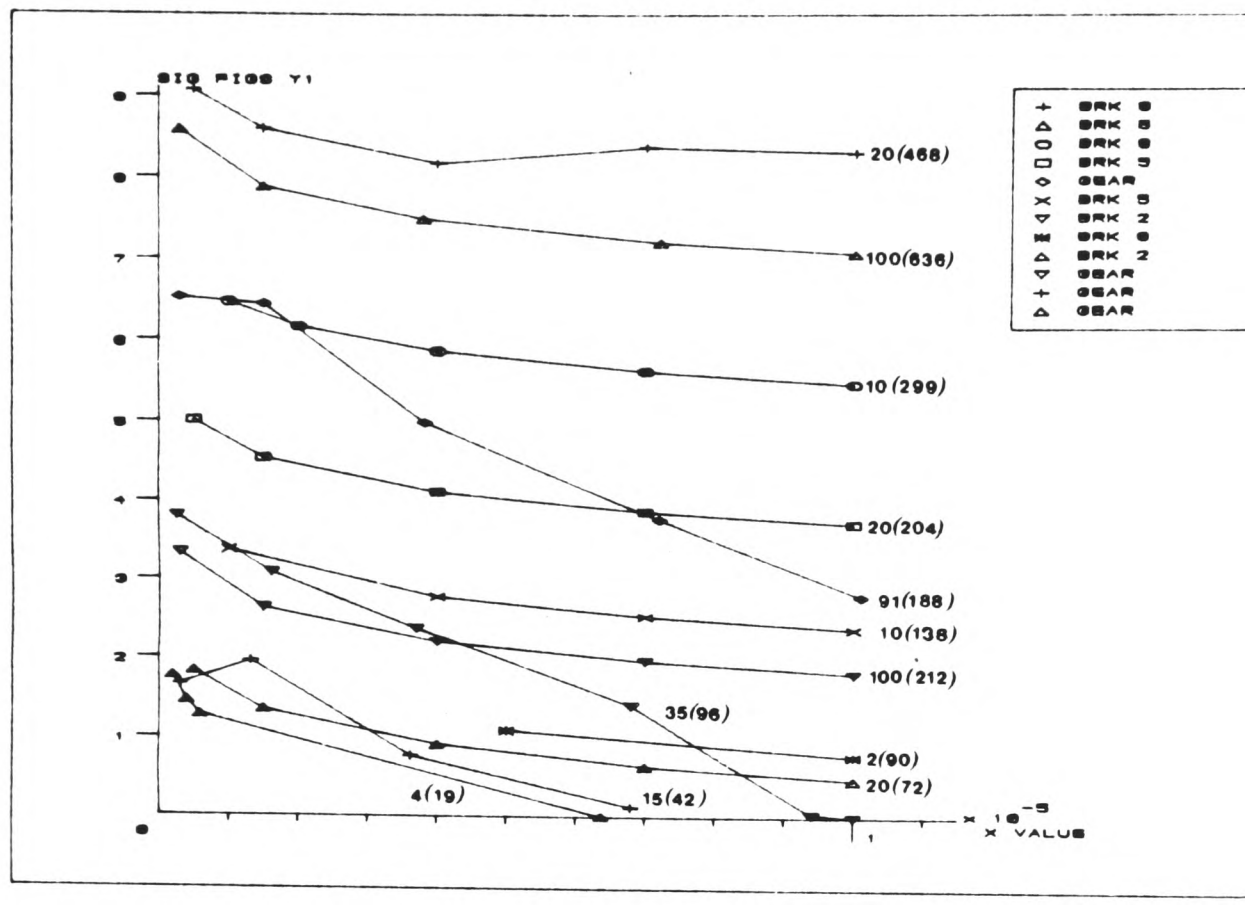


Figure 3.10 : Problem p3.2 over initial stage.

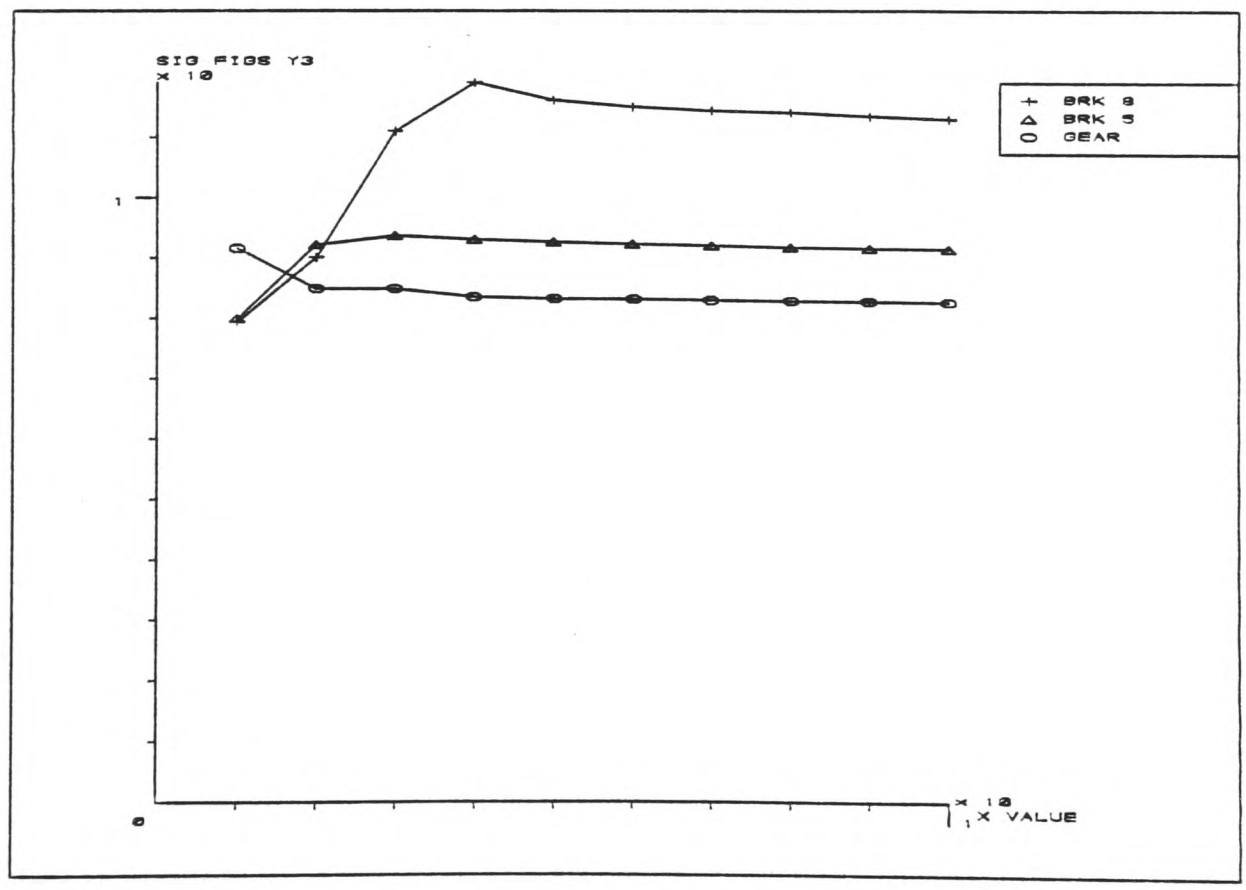


Figure 3.11 : Problem p3.2 over whole range.

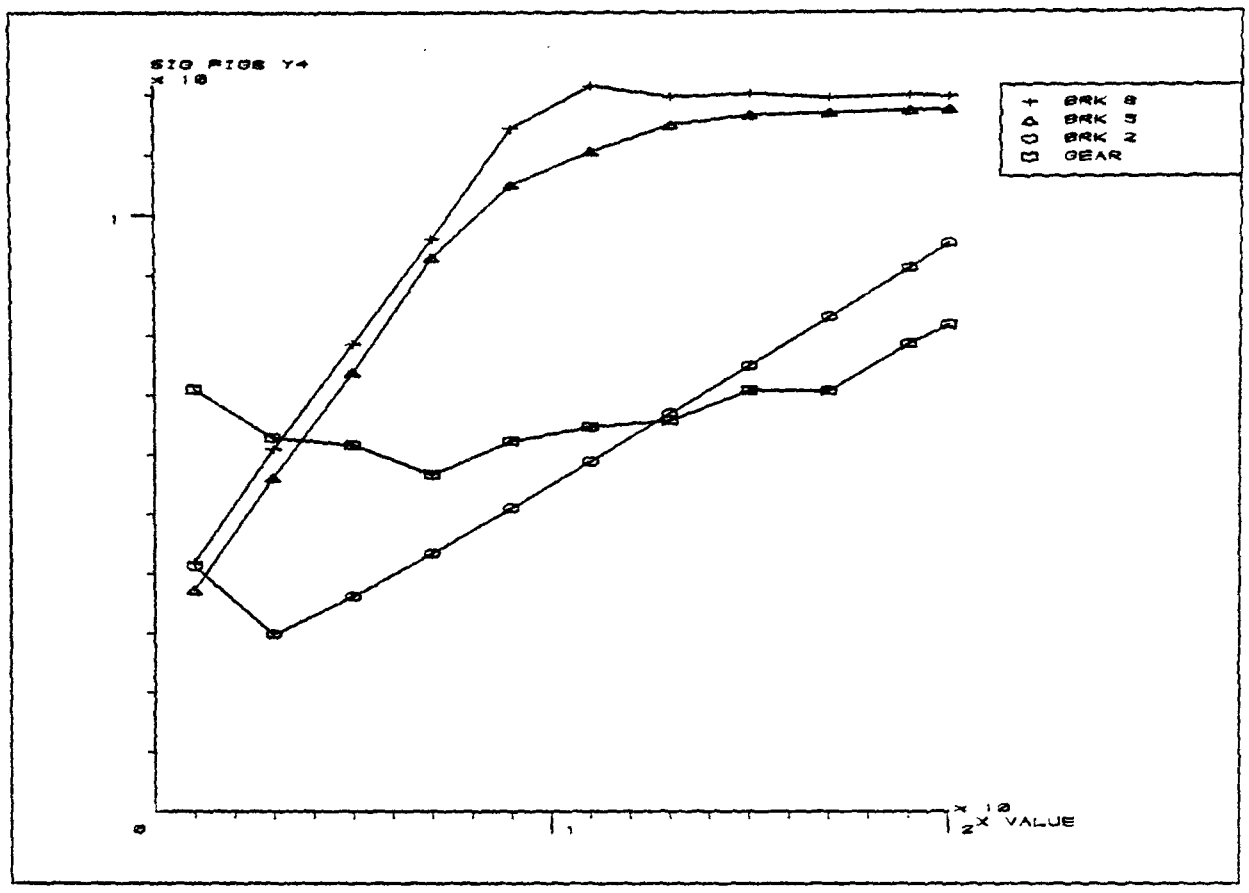


Figure 3.12 : Problem p3.3 over whole range.

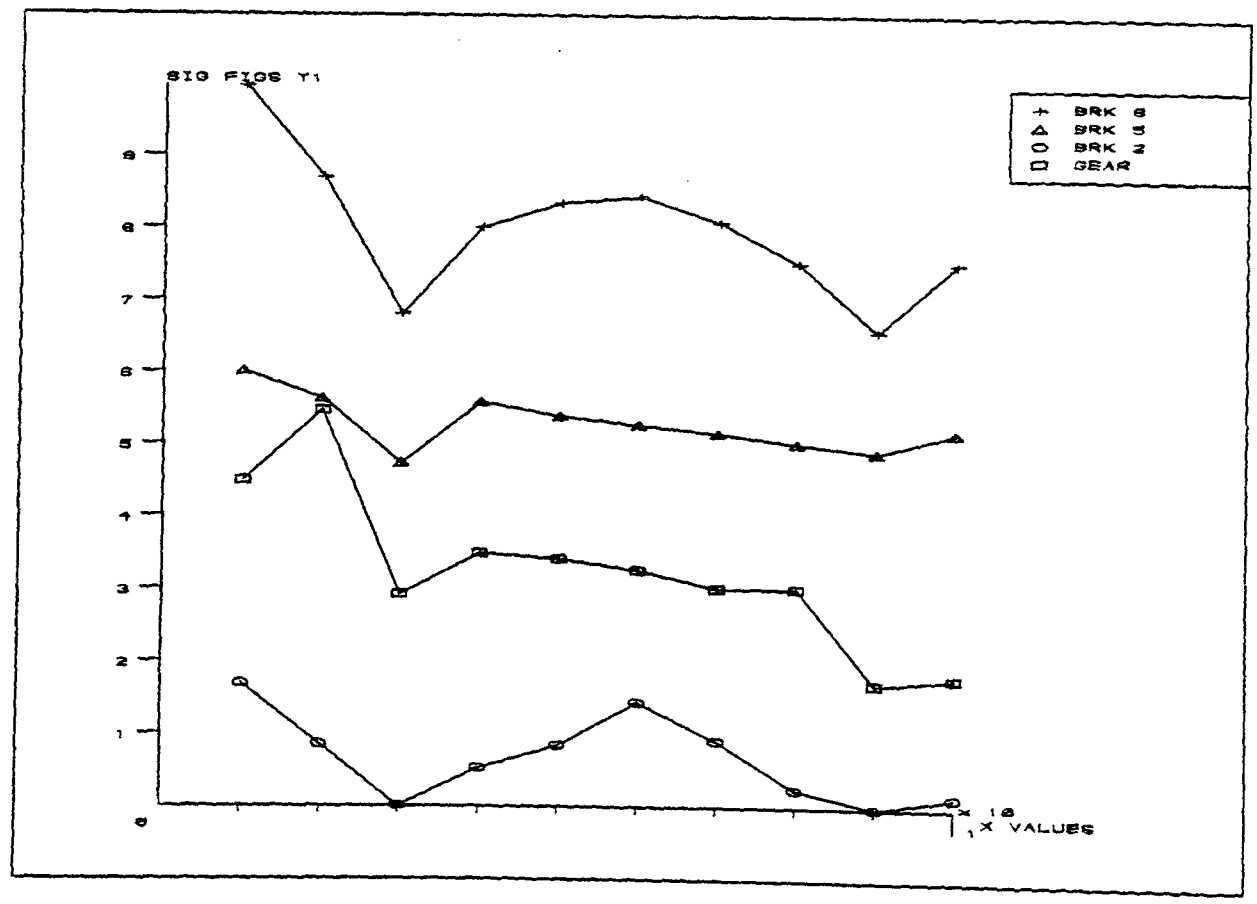


Figure 3.13 : Problem p3.4 over whole range.

Chapter 4 : ERROR CONTROL

In chapter 3 the potential of BRK methods for solving stiff systems of ODEs was demonstrated using fixed step mode. However, some means of controlling the error of the method must be derived to produce an efficient algorithm.

The global error, the difference between the true solution and the numerical one at any given point, cannot generally be determined. Thus the usual measure of control is the local truncation error, ie. the error committed in one step of the method assuming that no errors have previously been introduced. In fact the global error is a result of the local truncation errors, formed over all the previous steps, taken to construct the numerical solution, accumulated in a non-trivial manner together with round-off errors. In addition for an implicit method, since the implicit equations are never solved exactly, further errors are introduced. It can be shown that a bound on the local truncation error provides a corresponding bound on the global error, Lambert[1973].

This chapter examines a variety of techniques for estimating the principal local truncation error of a Runge-Kutta method and in particular BRK methods. By estimating the principal local truncation error the step size, h, can be adjusted automatically so that in some region of x where y(x) is changing rapidly h is kept small, while in regions where y(x) is changing slowly h is made large.

4.1 Embedding

Unfortunately the error term, given by the Taylor series expansion, is too complicated to be of any practical value, hence the local

truncation error must be evaluated numerically. The most commonly used estimation method for Runge-Kutta methods is that of embedding, Fehlberg[1970], Verner[1978].

This technique requires an s-stage method (explicit or implicit) of order p+1, (s,p+1) to have embedded within it a method of order p, (s,p). By embedding we mean that the (s,p) method uses the same function evaluations as the (s,p+1), ie. the same sampling points, but uses a different linear combination. Assuming that the solution at x_{n-1} is exact and no computational errors are introduced in computing the next step and two approximations are generated at $x_n = x_{n-1} + h$. Let these be y_n and y_n^* for the (s,p+1) and (s,p) methods respectively.

Clearly if $y(x_n)$ denotes the true solution at x_n, then the (s,p+1) method generates

$$y_n = y(x_n) + h^{p+2}\phi(y(x_n)) + O(h^{p+3}) \qquad (4.1)$$

where $\phi(y(x_n))$ is the principal error function. The (s,p) method will produce

$$y_n^* = y(x_n) + h^{p+1}\chi(y(x_n)) + O(h^{p+2}) \qquad (4.2)$$

where $\chi(y(x_n))$ is the principal error function of the (s,p) method. Subtracting (4.2) from (4.1) yields the difference between the two estimates

$$y_n - y_n^* = h^{p+1}\chi(y(x_n)) + O(h^{p+2}) \qquad (4.3)$$

and hence

$$h^{p+1}\chi(y(x_n)) = y_n - y_n^* \equiv d(h) \qquad (4.4)$$

Thus d(h) is an estimate of the local truncation error of the (s,p) method at x_n and this must be kept less than a fixed local error tolerence, TOL, at each step to maintain a bound on the global error. An absolute error test or a relative error test may be used. In the

latter case

$$T(h) = \frac{\|d(h)\|}{\|y_n\|} \qquad (4.5)$$

and it is T(h) which is controlled. When the step has been completed, the optimal step size, $h_{new} = \mu h$, to be attempted on the next step can be estimated. This is the step which would exactly satisfy the local error requirements, TOL. Assuming that the error changes slowly along the integration range then μ must be chosen such that

$$T(\mu h) = TOL \qquad (4.6)$$

Clearly from (4.4) and (4.5)

$$T(\mu h) = \mu^{p+1} T(h) \qquad (4.7)$$

and it follows that

$$\mu = c \left[\frac{TOL}{T(h)}\right]^{1/(p+1)} \qquad (4.8)$$

where c is introduced as a safety factor, usually taken as $c \in [0.8, 1.0]$.

Once μ has been estimated from (4.8) then the following step control policy is adopted:

a) If $\mu < 0.8$ the the step is rejected. The step to be attempted on the re-calculation is $h_{new} = \nu h$, with $\nu = Max[0.1, \mu]$, to disallow very large changes in the step size.

b) If $1 \geq \mu \geq 0.8$ the solution produced by the (s,p) method is accepted. However, it is normal practise to carry forward the solution obtained from the (s,p+1) method and this is the policy adopted here. The new step is, however, reduced by setting $h_{new} = \mu h$.

c) If $\mu > 1$ then the step is accepted and again the p+1th order solution is carried forward. The new step is set at $h_{new} = \nu h$, where ν is constrained by $\nu = Min[10.0, \mu]$, again to disallow large changes in

the step size.

This is a typical error control policy for explicit Runge-Kutta methods.

4.2 Inverse embedding

This is a similar idea to the embedding technique, section 4.1, but is valid for the BRK methods derived in chapter 2. An embedded pair (s,p+1) and (s,p) are required as before. Suppose the implicit equations are solved for the (s,p+1) method to produce y_n, (the scalar case is depicted in Figure 4.1), so that

$$E(y_n, y_{n-1}) = y_n - y_{n-1} - h\sum_{i=1}^{s} c_i k_i = 0 \qquad (4.9)$$

The value of y_n together with the corresponding k_i values will in general not satisfy the (s,p) equation, ie.

$$E^*(y_n, y_{n-1}) = y_n - y_{n-1} - h\sum_{i=1}^{s} c_i^* k_i \neq 0 \qquad (4.10)$$

We could of course solve the (s,p) method equations to produce y_n^* giving

$$E^*(y_n^*, y_{n-1}) = y_n^* - y_{n-1} - h\sum_{i=1}^{s} c_i^* k_i^* = 0 \qquad (4.11)$$

but this will require a great deal more function evaluations to produce the k_i^* values which correspond to y_n^* instead of y_n. It will also mean estimating $\partial E^*/\partial y_n$. Alternatively for some vector y_{n-1}^* we have

$$E^*(y_n, y_{n-1}^*) = y_n - y_{n-1}^* - h\sum_{i=1}^{s} c_i^* k_i = 0 \qquad (4.12)$$

Let

$$y_{n-1}^* - y_{n-1} = \alpha$$

and

$$y_n^* - y_n = \beta$$

It is extremely easy to compute α but we really need β. From (4.11)

$$E^*(y_n^*, y_{n-1}) = E^*(y_n - \beta, y_{n-1})$$

$$= E^*(y_n, y_{n-1}) - \frac{\partial E^*}{\partial y_n}(y_n, y_{n-1})\beta + \text{HOT}$$

$$= 0$$

Ignoring the higher order terms

$$\beta \approx \left[\frac{\partial E^*}{\partial y_n}\right]^{-1} E^*(y_n, y_{n-1}) \qquad (4.13)$$

From (4.10) and (4.12)

$$\alpha = E^*(y_n, y_{n-1})$$

and

$$\beta \approx \left[\frac{\partial E^*}{\partial y_n}\right]^{-1} \alpha \qquad (4.14)$$

Thus β can be computed from α provided an estimate for $\partial E^*/\partial y_n$ is available.

4.3 Richardson extrapolation

One error control policy commonly used is that of Richardson extrapolation (halving or doubling). The integration from x_{n-1} to x_n is performed twice with the same order method. A step of 2h is taken and compared with a solution computed by taking two steps of size h.

Let the solution obtained by the 2h step be denoted by y_n^* and the solution by the two h steps by $y_{n-\frac{1}{2}}$ and y_n. Thus using $y(x_n)$ as before

$$y_n^* = y(x_n) - (2h)^{p+1}\phi(y(x_n)) + O(h^{p+2}) \qquad (4.15a)$$

and

$$y_n = y(x_n) - h^{p+1}\phi(y(x_n)) + O(h^{p+2}) \qquad (4.15b)$$

Hence, subtracting (4.15b) from (4.15a),

$$h^{p+1}\phi(y(x_n)) = \frac{y_n^* - y_n}{1 - 2^{p+1}} \equiv d(h) \qquad (4.16)$$

giving an estimate of the local truncation error, d(h). Using a similar approach to the embedding technique the optimal step to be attempted next can be estimated by

$$\mu = c \left[\frac{TOL(2^{p+1} - 1)}{T(h)}\right]^{1/(p+1)} \qquad (4.17)$$

where T(h) is as (4.5). The step to be attempted next, h_{new}, is then computed in the same way as with the embedding technique. The solution that is carried forward is the solution obtained by the two h steps, as this should be the most accuate.

There are two main drawbacks with this technique. The first is the amount of work required to perform one successful step ie. the step is performed twice with different step sizes to produce an estimate for the local truncation error. The second is that an iteration matrix is required for both stages, ie. one for the 2h step and one for the two h steps. This clearly makes the operation expensive.

4.4 Implementation details

The obvious choice for an error control policy, for ERK methods is the embedding technique. This is straight forward to implement and is done so as described above.

BRK methods on the other hand have two plausible error control techniques, 4.2 or 4.3. Extensive testing of the two techniques indicated that the Richardson technique was more reliable and more efficient. Table 4.1 shows the results for a typical problem, viz. p3.4. Hence this is the error control policy adopted. The basic

implementation details are described below.

The order in which the integration steps are performed in this technique is of great importance. The 2h step must be performed first, as it is more likely to fail than the small h steps. The initial estimate for y_n^* is obtained by extrapolating the divided difference table updated after each successful step. When the 2h step has been successfully completed, the solution produced is used to update the divided difference table. This old divided difference table is, however, not over-written by the new one as a step failure would destroy the validity of the table. Therefore this new table is calculated and then stored separately. The initial approximation to $y_{n-\frac{1}{2}}$, the first h step, is obtained by interpolating the new divided difference table. Providing that the first h step is computed successfully the second can be performed. An estimate of the solution at x_n already exists from the 2h step, and this can be used as an initial approximation to the solution for the second h step. By using this initial approximation the iterations normally converge rapidly, usually in 1 or 2 iterations. The test (4.17) is then conducted to determine whether the step was successful and determine h_{new} for the next step.

4.5 Numerical results

BRK methods of orders 5 and 8 have been implemented in variable step mode and used to integrate the problems considered in chapter 3, viz. p3.1, p3.2, p3.3 and p3.4. Due to the lack of error control of the methods in chapter 3 the approach adopted here, to analyse the results, is somewhat different. The methods can now be controlled by specifying a local error tolerance, TOL, that must be satisfied at each step. It

is, however, not enough to specify some value of TOL and compare this with the results of GEAR using the same tolerance level. Both methods will use TOL as a local error tolerance but may actually control different quantities. Thus the accuracy actually attained by the method must be monitored. The approach taken in this chapter is to monitor the accuracy at the end of the integration range in terms of the number of accurate significiant figures obtained over the relevant component. The relevant components being the same as those used in chapter 3.

The results of problem p3.1 are tabulated in Table 4.2 and displayed graphically in Figure 4.2. Even though an expensive error control policy is being employed the BRK methods are able to compute a solution more efficiently than GEAR, with the 5th order method marginally outperforming the 8th.

With problem p3.2, Table 4.3, Figure 4.3, all methods are capable of producing results in a reasonable time when low accuracy is required. If more accuracy is requested, however, the 8th order method is best.

Problem p3.3 highlighted the necessity of an error control policy for BRK methods (Figure 3.12), Table 4.4 and Figure 4.4, show the effect of incorporating an error control policy into the 5th order method. It shows that a more uniform accuracy is maintained throughout the integration range. The results of Table 4.5 and Figure 4.5 show that the 8th order method is extremely inefficient on this problem in total contrast to problem p3.2. Clearly the high number of stages of the methods, viz. 12, place a severe restriction on it, especially as the problem has more state variables than the other three problems

considered.

When a problem with fewer state variables, problem p3.4, is considered the 8th order method again shows its supremacy, (Table 4.6 and Figure 4.6).

Method	$\log_{10}$ TOL	FE	JE	Steps	Maximum Step	Accuracy at x_{end}	CPU Time
Richardson	-3	38094	22	2176	7.8e-3	2.02	41.94
	-4	55674	16	3202	5.3e-3	3.00	61.00
	-5	79980	18	4612	3.7e-3	3.78	87.70
	-6	122796	4	10002	1.0e-3	5.52	146.12
	-8	221496	6	12720	1.0e-3	6.03	242.00
	-9	310818	12	17858	6.9e-4	6.67	340.00
Inverse	-3	32852	21	2525	1.8e-2	2.46	28.64
Fehlberg	-4	51433	30	3982	1.1e-2	3.12	44.87
	-5	81439	34	6435	7.6e-2	4.05	72.02
	-6	142765	43	10537	4.8e-3	4.68	124.33
	-7	267409	45	14487	3.0e-3	5.03	218.30
	-8	402637	60	22293	1.9e-3	6.05	329.10
	-9	603211	67	35515	1.1e-3	6.81	503.70

Table 4.1 : Comparison of error control devices

Method & Order	$\log_{10}$ TOL	FE	JE	Steps	Maximum Step	Accuracy at x_{end}	CPU Time
BRK 8	-3	1752	18	18	9.6e-1	5.99	1.97
	-4	1908	24	24	8.5e-1	7.21	2.15
	-6	2916	10	54	8.7e-1	7.93	3.60
	-7	3336	12	58	8.4e-1	9.14	4.09
	-8	3000	10	76	2.3e-1	10.66	3.88
	-9	4020	7	104	1.9e-1	11.11	5.26
BRK 5	-2	972	21	24	8.0e-1	4.66	1.36
	-4	858	11	34	3.1e-1	6.66	1.29
	-6	1368	12	60	8.5e-1	7.74	2.11
	-7	1920	15	86	4.1e-1	8.48	2.99
	-8	2443	10	150	6.1e-2	9.38	4.04
	-9	3349	12	228	4.7e-2	11.73	7.95
GEAR	-5	234	21	130	6.4e-1	3.64	1.78
	-6	293	25	169	4.0e-1	4.72	2.39
	-7	424	30	241	3.6e-1	5.57	3.43
	-8	462	34	300	2.6e-1	6.19	4.08
	-9	619	43	414	1.7e-1	6.99	5.66
	-10	809	48	576	1.1e-1	7.83	7.57
	-11	1056	61	809	9.0e-2	8.59	10.31
	-12	1415	76	1132	6.0e-2	9.38	14.14

Table 4.2 : Comparison of problem p3.1 (Figure 4.2)

Method & Order	$\log_{10}$ TOL	FE	JE	Steps	Maximum Step	Accuracy at x_{end}	CPU Time
BRK 8	-2	1212	11	16	7.0e-1	1.39	1.84
	-3	1200	16	16	1.8e 0	3.53	1.77
	-7	1488	21	20	9.8e-1	4.80	2.19
	-8	2052	28	26	8.3e-1	5.70	3.03
	-9	3133	30	48	6.1e-1	7.90	4.79
	-10	3589	29	52	8.4e-1	8.10	5.50
BRK 5	-3	726	22	20	1.8e 0	1.79	1.23
	-4	900	20	42	2.4e-1	2.53	1.66
	-6	1254	12	90	1.1e-1	4.08	2.55
	-7	1657	10	132	7.7e-2	4.58	3.47
	-8	2239	8	196	5.3e-2	5.34	4.80
	-9	3446	13	290	3.5e-2	6.24	7.34
	-10	4802	13	428	2.4e-2	7.10	10.36
GEAR	-6	180	13	102	2.6e-1	1.55	1.49
	-7	231	17	140	1.9e-1	2.86	1.97
	-8	302	20	195	1.5e-1	3.86	2.74
	-9	382	22	274	1.0e-1	4.23	3.70
	-10	520	29	385	6.8e-2	4.89	5.16
	-11	722	40	555	5.1e-2	5.44	7.29
	-12	1028	53	813	3.4e-2	6.22	10.60

Table 4.3 : Comparison on problem p3.2 (Figure 4.3)

Method & Order	$\log_{10}$ TOL	FE	JE	Steps	Initial Step	Maximum Step	CPU Time
BRK 5	-3	2136	38	80	1.0e-4	1.1e 0	3.47
GEAR	-6	383	23	206	4.8e-6	1.0e 0	4.40

Table 4.4 : Comparison on problem p3.3 (Figure 4.4)

Method & Order	Log_{10} TOL	FE	JE	Steps	Maximum Step	Accuracy at x_{end}	CPU Time
BRK 8	-1	3936	51	34	1.4e 0	5.74	5.37
	-3	3324	34	42	1.2e 0	6.99	4.71
	-5	3684	29	60	9.7e-1	9.46	5.47
BRK 5	-2	1854	35	72	9.9e-1	6.99	2.98
	-4	2814	28	140	9.8e-1	8.93	5.00
	-5	4722	33	346	1.0e 0	9.87	9.33
GEAR	-2	129	12	50	3.0e 0	4.08	1.12
	-3	193	14	80	2.7e 0	4.96	1.77
	-4	276	22	124	3.1e 0	5.60	2.70
	-5	278	16	148	9.3e-1	6.09	3.14
	-7	479	27	274	3.9e-1	9.50	5.80
	-9	1032	60	578	5.3e-1	9.81	12.18
	-10	1146	62	732	3.9e-1	10.47	14.66

Table 4.5 : Comparision on problem p3.3 (Figure 4.5)

Method & Order	Log_{10} TOL	FE	JE	Steps	Maximum Step	Accuracy at x_{end}	CPU Time
BRK 8	-3	23484	12	634	2.3e-2	1.24	21.08
	-4	31260	12	838	1.3e-2	2.19	27.93
	-5	32964	10	958	1.2e-2	2.74	30.16
	-6	52668	22	1436	9.6e-3	4.20	46.88
	-7	68352	16	1956	7.1e-3	5.34	60.81
	-8	82740	12	2982	3.3e-3	6.77	77.16
	-9	108024	25	3310	3.1e-3	7.20	92.23
BRK 5	-3	38094	22	2176	7.8e-3	2.02	41.94
	-4	55674	16	3202	5.3e-3	3.00	61.00
	-5	79980	18	4612	3.7e-3	3.78	87.70
	-6	122796	4	10002	1.0e-3	5.52	146.12
	-8	221496	6	12720	1.0e-3	6.03	242.00
	-9	310818	12	17858	6.9e-4	6.67	340.00
GEAR	-7	11481	628	9794	3.1e-3	1.80	99.53
	-8	15985	797	14020	2.6e-3	2.59	139.54
	-9	23013	1115	20402	1.8e-3	3.41	202.76
	-10	32907	1577	29296	1.0e-3	4.28	290.63
	-11	48742	2343	43505	7.9e-4	5.11	426.31

Table 4.6 : Comparison on problem p3.4 (Figure 4.6)

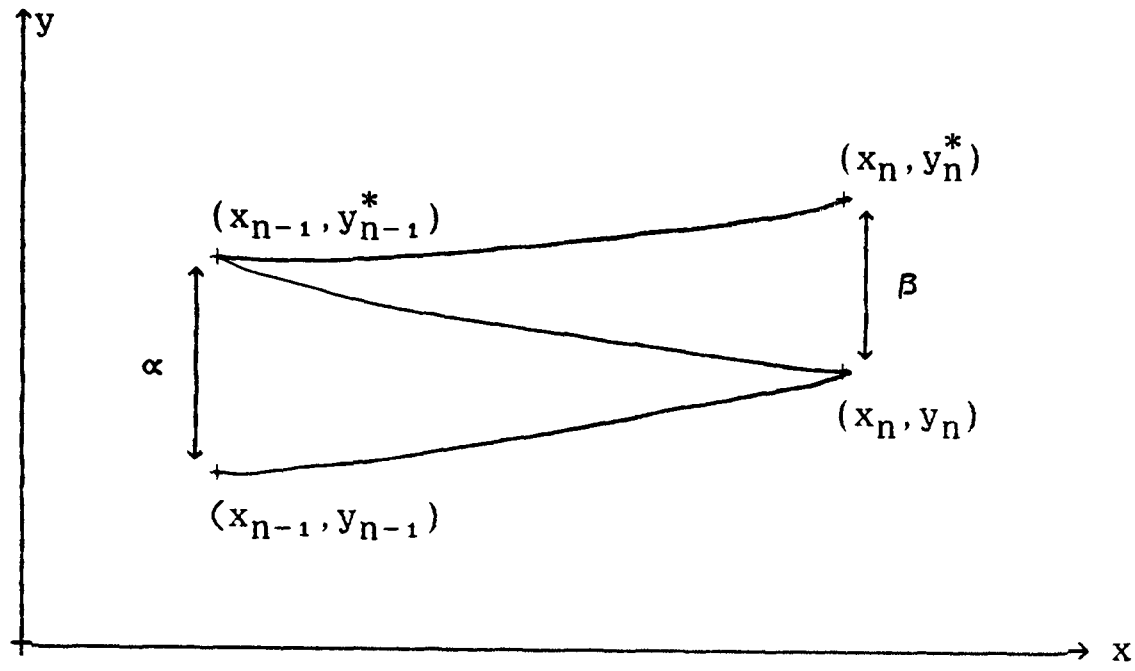


Figure 4.1 : Node layout for inverse embedding method (scalar case)

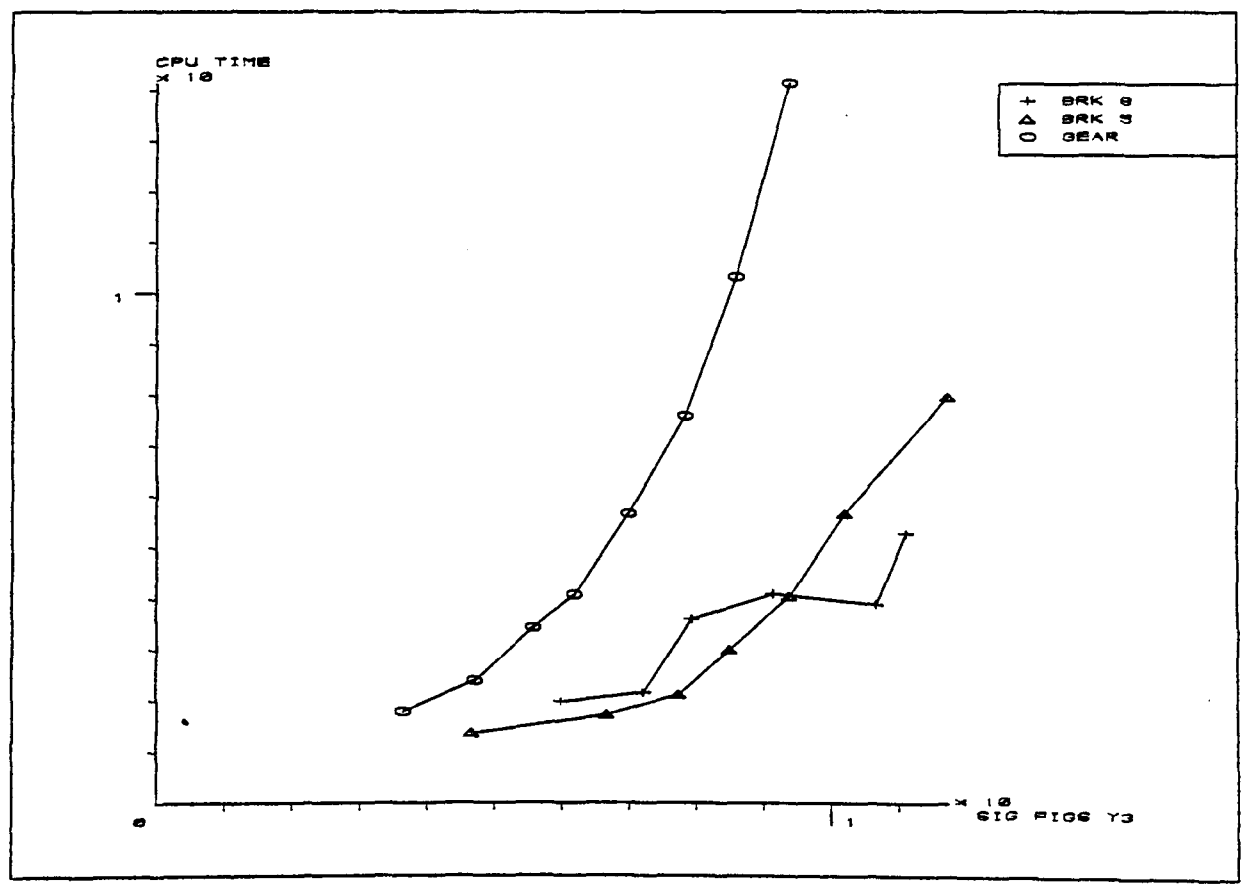


Figure 4.2 : Problem p3.1

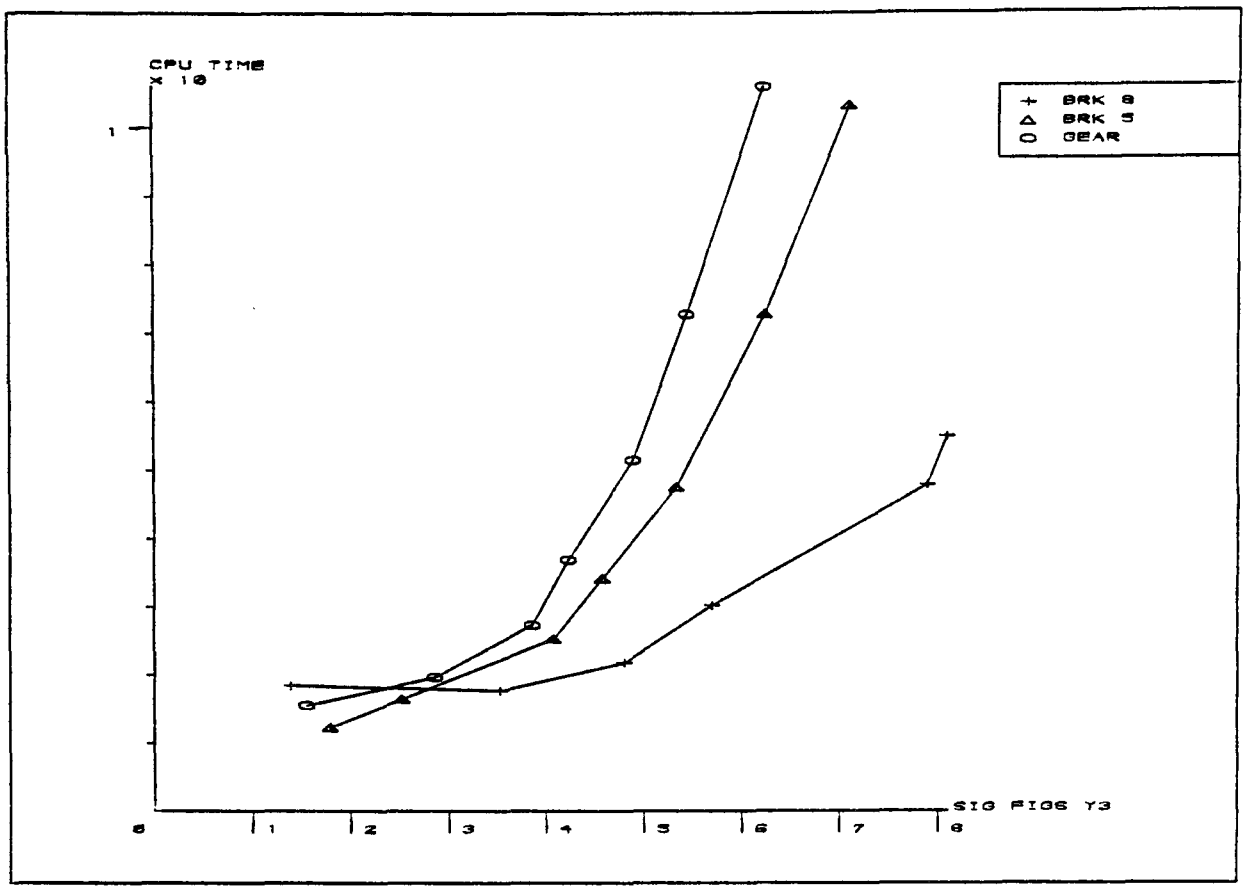


Figure 4.3 : Problem p3.2.

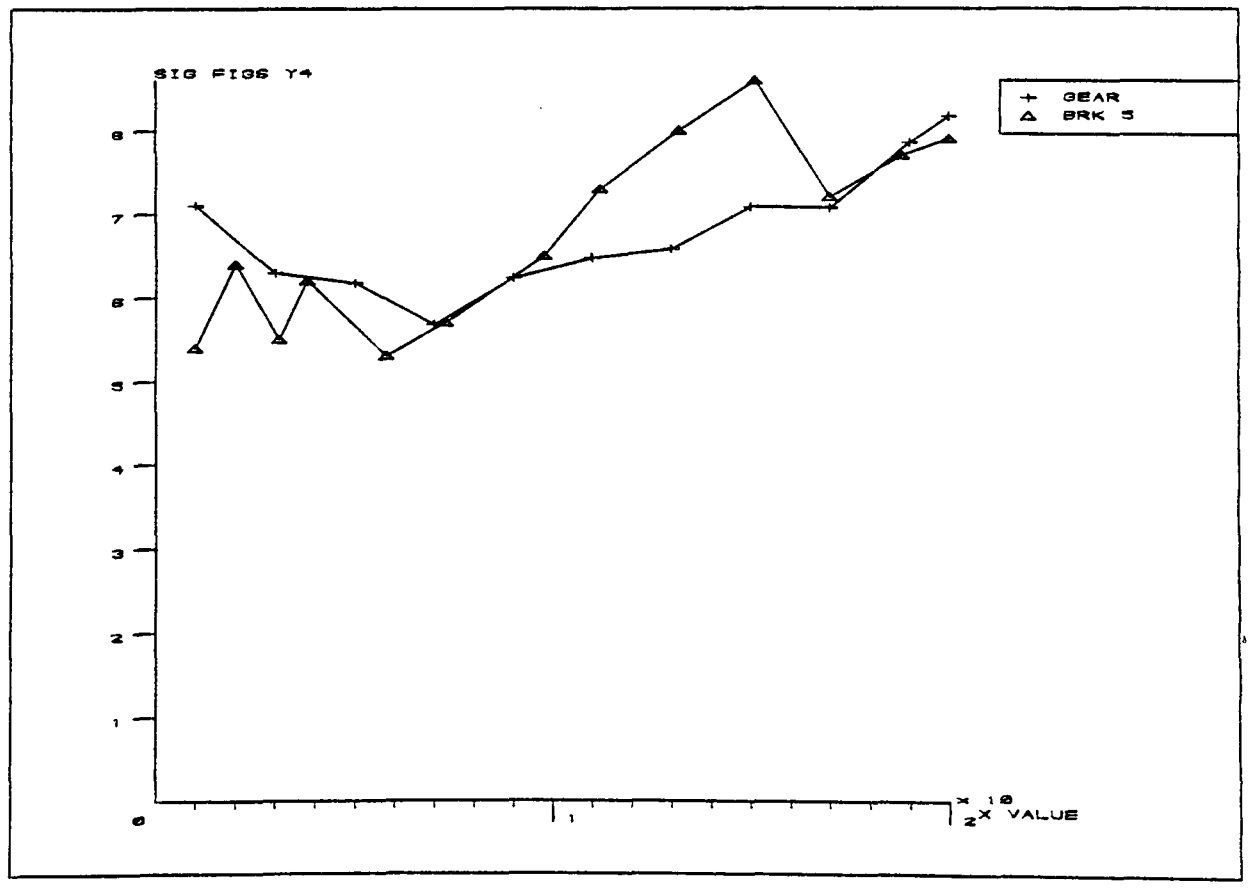


Figure 4.4 : Problem p3.3 over whole range.

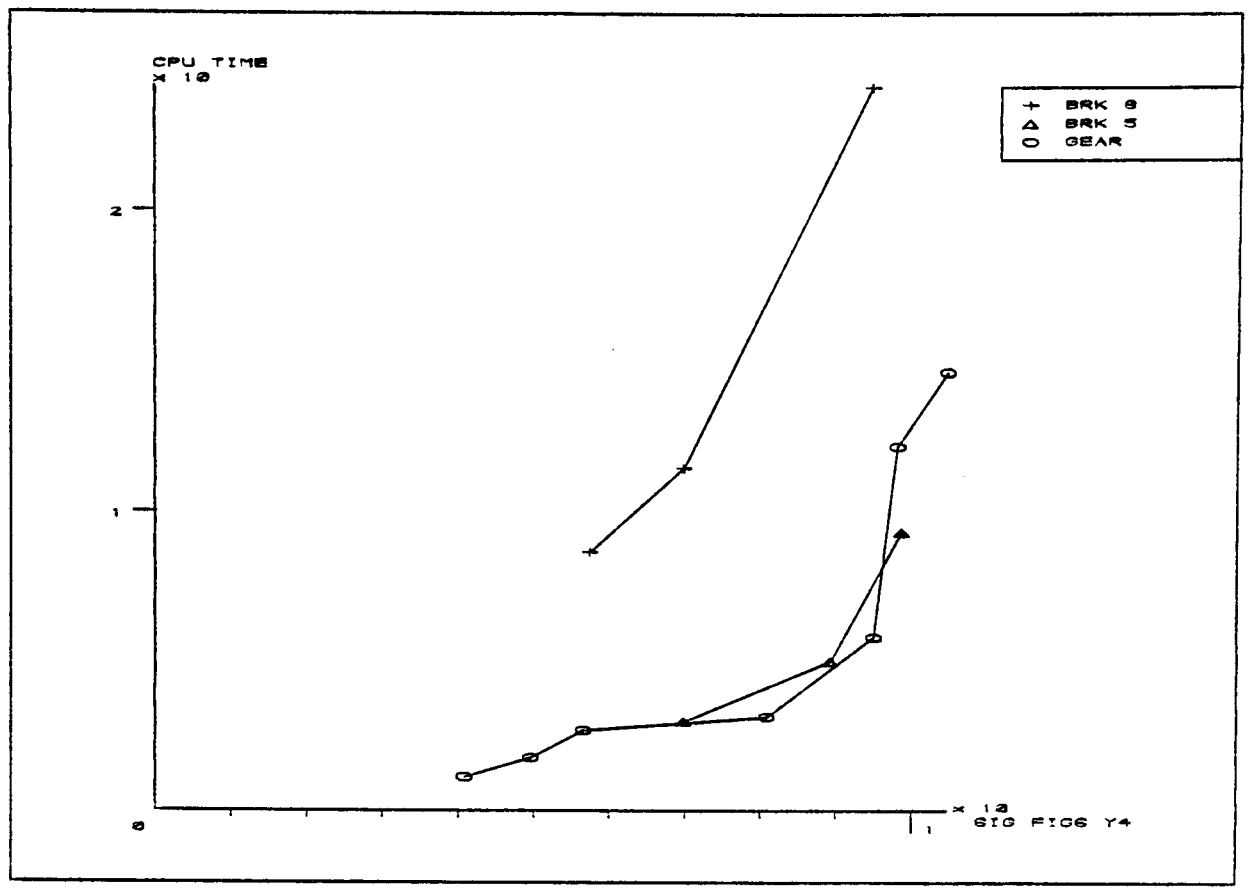


Figure 4.5 : Problem p3.3.

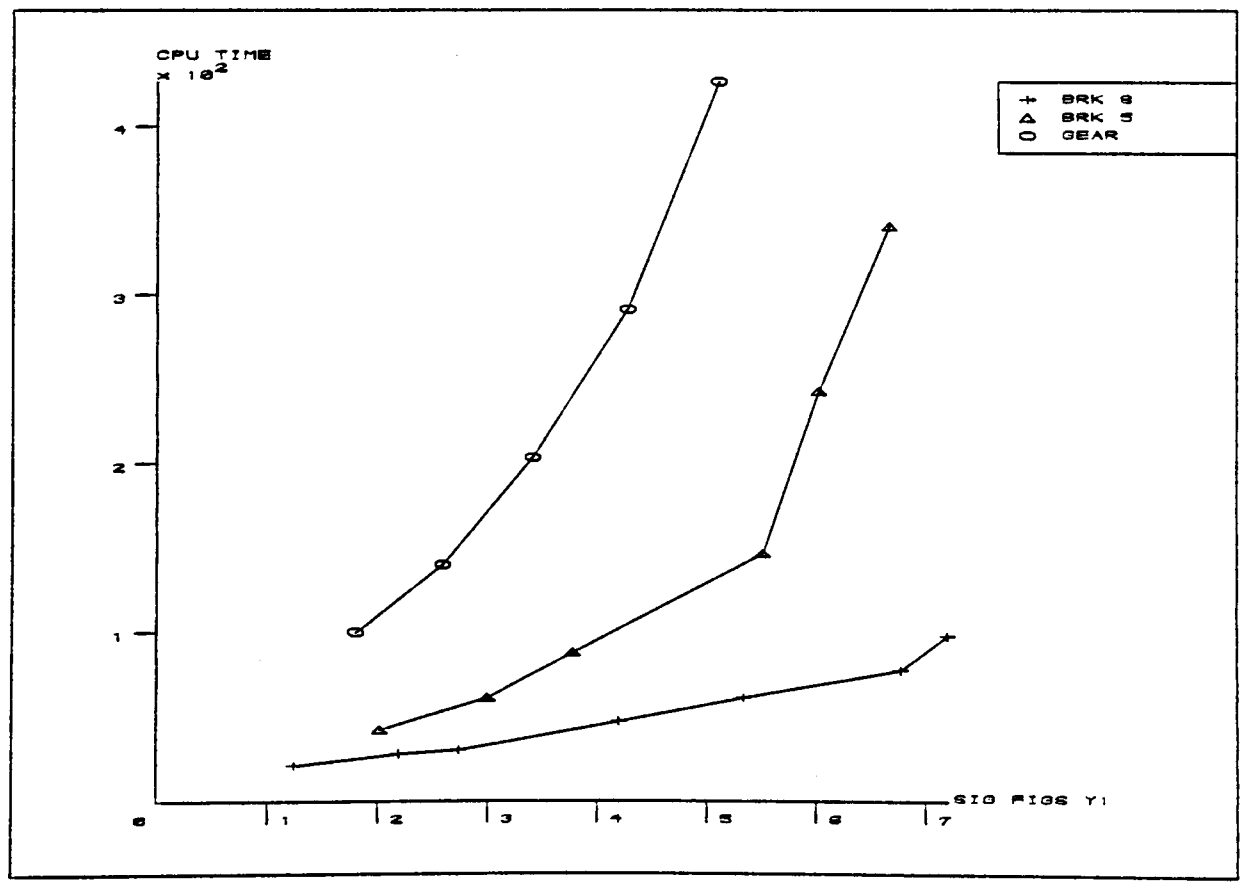


Figure 4.6 : Problem p3.4.

Chapter 5 : PROBLEMS ASSOCIATED WITH BRK METHODS

All numerical methods used to solve ODEs will be inefficient for some class of problem. Thus Adams methods are hopelessly inefficient at solving stiff systems and BDF methods are poor when applied to highly oscillatory problems. Not surprisingly BRK methods are inefficient for some problems.

This chapter sets out the deficiencies of BRK methods and some of the routes that have been investigated in the search for a remedy.

5.1 Singular iteration matrix

When integrating a well-behaved system of ODEs we expect the iteration matrix associated with the solution of the implicit equations to be non-singular. However, in some cases, due to the finite precision of the calculations, singular or nearly singular approximations to the Jacobian may be produced. This normally happens only very rarely. Any isolated occurrence of it can be overcome by simply changing the step size. A step increase is not advocated as then the accuracy requirements may not be met, thus the step must be reduced. With BRK methods the iteration matrix is computed directly from the residual vector, $\epsilon(h)$, by perturbing the y_n values and recomputing $\epsilon(h)$, section 3.5.

Consider the linear system

$$\frac{dy_1}{dx} = y_2$$

$$\frac{dy_2}{dx} = -10^{\alpha}y_1 - (10^{\alpha} + 1)y_2 \qquad x\epsilon[0,1] \qquad (p5.1)$$

with initial conditions

$$y_1(0) = 2 \qquad y_2(0) = -10^{\alpha}$$

and α is a positive integer. The Jacobian matrix associated with p5.1 is

$$J = \begin{bmatrix} 0 & 1 \\ -10^{\alpha} & -(10^{\alpha}+1) \end{bmatrix} \tag{5.1}$$

which is clearly non-singular.

From chapter 3 the iteration matrix asscoiated with an s-stage BRK method, for a linear system, has the form

$$M = \sum_{i=o}^{s} d_i(-hJ)^i \tag{5.2}$$

A 3-stage 3rd order BRK method will thus have an iteration matrix comprising

$$I - hJ + (hJ)^2/2 - (hJ)^3/6 \tag{5.3}$$

Now clearly J^2 and J^3 for problem p5.1 can be constructed as

$$J^2 = \begin{bmatrix} -10^{\alpha} & -10^{\alpha}-1 \\ 10^{2\alpha}+10^{\alpha}+1 & 10^{2\alpha}+10^{\alpha}+1 \end{bmatrix}$$

and (5.4)

$$J^3 = \begin{bmatrix} 10^{2\alpha}+10^{\alpha} & 10^{2\alpha}+10^{\alpha}+1 \\ -10^{3\alpha}-10^{2\alpha}-10^{\alpha} & -10^{3\alpha}-10^{2\alpha}-10^{\alpha}-1 \end{bmatrix}$$

For a large value of α and moderately sized h

$$M \approx -h^3J^3/6 \tag{5.5}$$

which may be computed as singular since the second column is virtually the same as the first and hence the system of equations cannot be solved. This is a problem that is inherent to all MIRK based methods, Singhal[1980]. The situation degenerates as the order of the method increases. Even though the implementation discussed in chapter 3 does not compute J directly, the approximation to the iteration matrix, M, may be computed as singular.

This severe drawback of MIRK methods may make them unsuitable as an integrator for a class of simple linear ODEs. However, various schemes to alleviate this problem are discussed in the following section.

It is important to note that it is not the eigenvalues that cause the problems since systems with exactly the same eigenvalues as p5.1 can be solved very efficiently. Thus it is untrue to say that BRK methods cannot solve stiff systems but rather that they are inefficient at solving some stiff systems. For example the problem

$$\frac{dy_1}{dx} = -10^6 y_1$$

$$\frac{dy_2}{dx} = -y_2 \qquad \text{(p5.2)}$$

has the same eigenvalues as problem p5.1, yet BRK methods are able to integrate this problem without any difficulties.

5.1.2 Approximate factorization

Following an approach proposed by Singhal[1980], the iteration matix can be factorized as

$$M = (I-\nu_1 hJ)(I-\nu_2 hJ) \ . \ . \ . \ (I-\nu_s hJ) \qquad (5.6)$$

for an s-stage method. Singhal[1980] shows that a more efficient scheme can be produced by taking $\nu_i \equiv \nu$, i=1(1)s. Thus M is approximated by

$$M \simeq (I - \nu hJ)^s \qquad (5.7)$$

The solution of which will require one Jacobian evaluation, one LU decomposition and s back substitutions. Clearly ν must be chosen so that the approximation to the iteration matrix produces a matrix that ensures that the rate of convergence of the Newton process is less than one , ie. $\rho(M) < 1$, and hopefully $\rho(M) << 1$. Singhal[1980] found

values of ν for MIRK methods of order 2, 3 and 4 such that $\rho(M)$ was less than one.

This idea can be extended to allow two free parameters and hence giving the scheme more flexibility. Consider

$$M = (\alpha I + \beta hJ)^s \qquad (5.8)$$

as an approximation to the iteration matrix. The free parameters can be found by a minimax process. Consider for simplicity, a linear approximate factorization for the 2-stage 2nd order BRK method. Application of Newton's method gives

$$y_{n+1}^{(k+1)} = A\, y_{n+1}^{(k)} + b \qquad (5.9)$$

where

$$A = I - (\alpha I + \beta hJ)^{-2}(I - hJ + (hJ)^2/2)$$

$$b = (\alpha I + \beta hJ)^{-2} y_n$$

To ensure that the approximation is satisfactory $\rho(A)$ must be less than 1, ie. the largest eigenvalue of A cannot be greater than one. Now the eigenvalues of A can easily be found by replacing J by λ_i in (5.9), thus A has eigenvalues

$$1 - (\alpha + \beta h\lambda_i)^{-2}(1 - h\lambda_i + (h\lambda_i)^2/2) \qquad (5.10)$$

These eigenvalues must be minimized with respect to α and β over some region of the $h\lambda(=q)$ complex plane. Singhal[1980] considered the approximation to the iteration matrix, (5.7), over the complete left-hand half plane, ie. $Re(q) \leq 0$. By ensuring that (5.10), (with $\alpha \equiv 1$), is analytic in this region then the maximum modulus theorem was used to show that the maximum value is attained on the boundary of the region, ie. $Re(q) = 0$. The approach taken here is somewhat different. The problems that create severe difficulties for BRK methods have real eigenvalues on the negative axis. Thus we perform our minimization solely over the real negative axis. Hence we require

$$\underset{\alpha,\beta}{\text{Min}} \quad \underset{q\in(-\infty,0]}{\text{Max}} \left| 1 - \frac{(1-q+q^2/2)}{(\alpha+\beta q)^2} \right| \qquad (5.11)$$

Put

$$\mu = 1/(\alpha + \beta q) \Rightarrow q = (1-\alpha\mu)/\beta\mu$$

Thus, as there is a 1-1 relationship between q and μ, the problem can be rewritten to allow a finite range of μ to be inspected, ie.

$$\underset{\alpha,\beta}{\text{Min}} \quad \underset{\mu\in[0,1/\alpha]}{\text{Max}} \; | \epsilon(\mu) | \qquad (5.12)$$

where

$$\epsilon(\mu) = \frac{-2(\alpha+\beta)^2\mu^2 - 2(\alpha+\beta)\mu + 1 + 2\beta^2}{2\beta^2} \qquad (5.13)$$

for a 2-stage 2nd order BRK method. Clearly $(\alpha+\beta q)$ must be non-zero to ensure that the function is analytic. We take α as positive and β negative to ensure this.

The NAG subroutine E04CGF was used to determine the values of α and β to minimize $\epsilon(\mu)$. This routine finds the minimum value of a function of N independent variables using function values only. The function that E04CGF minimizes is the maximum of $\epsilon(\mu)$ over the finite interval $[0,1/\alpha]$, ie. a minimax problem. This maximum can be determined by computing $\epsilon(0)$, $\epsilon(1/\alpha)$ and $\epsilon(\mu)$ at the turning points given by $\epsilon'(\mu) = 0$ such that $\mu\in[0,1/\alpha]$. The maximum of the moduli of these value of $\epsilon(\mu)$ gives the function values for E04CGF and the minimum of those produces the required amplification factor.

This idea can be extended further by using an approximation to the iteration matrix

$$M \simeq \frac{1 + \delta_1 hJ + \delta_2 (hJ)^2 + \ldots + \delta_m (hJ)^m}{(\alpha I + \beta hJ)^{2+m}} \qquad (5.14)$$

for the 2-stage 2nd order BRK method. The process (5.14) can however,

be made more efficient by rewriting and implementing it in the following form

$$\frac{1}{(\alpha I+\beta hJ)^2} + \frac{h_1}{(\alpha I+\beta hJ)^3} + \ldots + \frac{h_m}{(\alpha I+\beta hJ)^{2+m}} \qquad (5.15)$$

This has m+2 free parameters which can be adjusted to reduce the amplification factor. This involves solving

$$\underset{\substack{\alpha,\beta \\ h_1,..,h_m}}{\text{Min}} \quad \underset{q\in(-\infty,0]}{\text{Max}} \left| 1 - \frac{(1-q+q^2/2)}{(\alpha+\beta q)^2+\ldots+h_m(\alpha+\beta q)^{2+m}} \right| \qquad (5.16)$$

for the unknowns α, β, h_1, . . ., h_m, where the constraints are as above.

By extending this idea still further a quadratic factorization can be considered. This has the general form for a 6-stage 5th order BRK method of

$$(\alpha + \beta hJ + \gamma(hJ)^2)^3 \qquad (5.17)$$

Like the linear case a number of correction terms can be added to enhance the scheme, ie

$$\frac{1 + \delta_1 hJ + \delta_2(hJ)^2 + \ldots + \delta_{2m}(hJ)^{2m}}{(\alpha I + \beta hJ + \gamma(hJ)^2)^{m+6}} \qquad (5.18)$$

which can be written and computed as

$$\frac{1}{(\alpha I+\beta hJ+\gamma(hJ)^2)} + \ldots + \frac{h_m}{(\alpha I+\beta hJ+\gamma(hJ)^2)^{6+m}} \qquad (5.19)$$

for efficiency.

The amplification factor can be determined by using a similar idea to that mentioned in the previous section, except that with the quadratic case there is one extra parameter to solve for, viz. γ.

Similarly these ideas can be extended to higher order BRK methods. When

an approximate factorization is used for a 5th order 6-stage method the amplification factors can be very good. The amplification factors found for the approximate quadratic factorization of a 6-stage 5th order method are shown in Table 5.1. If more than four correction terms are added then the time required to perform the back substitution stage far outweighs the time saved in reducing the amplification factor.

5.1.2 Application of approximate factorization

The general code developed here uses orders 3 and 5 and hence an approximate factorization for the 5th order method is required. The form of the iteration matrix for a general 6-stage 5th order method is

$$M = I - hJ + (hJ)^2/2 - (hJ)^3/6 + (hJ)^4/24 - (hJ)^5/120 + (hJ)^6\rho \qquad (5.20)$$

where ρ depends upon the particular 5th order method being employed. Thus a linear approximate factorization of (5.20) would be

$$M \simeq (\alpha I + \beta hJ)^6 \qquad (5.21)$$

which can be formed cheaply and the LU decomposition successfully performed. However, it turns out that this approach cannot produce the desired results when it is used on high order methods. There are two stages in the process, viz. calculating the residual vector and performing the back substitutions. Simple numerical tests shows that the residual vector, used to form the iteration matrix, is correct but that the back substitution stage of the process goes wrong.

Consider problem p5.1 with $\alpha = 6$, then the eigenvalues of J can be found by forming the characteristic equation

$$\lambda^2 + \lambda(10^6+1) + 10^6 = 0 \qquad (5.22)$$

which clearly has roots at -10^6 and -1. The -10^6 egienvalue will be

referred to as the dominant eigenvalue. The corresponding eigenvectors can be formed from each eigenvalue, respectively as $[-10^{-6},1]^T$ and $[-1,1]^T$. Let these vectors be denoted by V_1 and V_2, respectively.

Defining the residual vector for a general s-stage BRK method as

$$r = y_{n+1} - y_n - h\sum_{i=1}^{s} c_i k_i \qquad (5.23)$$

then for a 6-stage 5th order method applied to a linear problem it is

$$r = [I - hJ + (hJ)^2/2 - (hJ)^3/3! + \ldots + \rho(hJ)^6]y^o_{n+1} - y_n \qquad (5.24)$$

Clearly y^o_{n+1} and y_n can be formed by taking a linear combination of the eigenvectors of J, ie.

$$y^o_{n+1} = a\begin{bmatrix}-10^{-6}\\ 1\end{bmatrix} + b\begin{bmatrix}-1\\ 1\end{bmatrix} \qquad (5.25a)$$

$$y_n = c\begin{bmatrix}-10^{-6}\\ 1\end{bmatrix} + d\begin{bmatrix}-1\\ 1\end{bmatrix} \qquad (5.25b)$$

hence the residual vector, (5.24), can be expressed as

$$r = a\{1 + h10^6 + (h10^6)^2/2 + \ldots + \rho(h10^6)^6\}\begin{bmatrix}-10^{-6}\\ 1\end{bmatrix} + b\{1 + h + h^2/2 + \ldots + \rho h^6\}\begin{bmatrix}-1\\ 1\end{bmatrix} + c\begin{bmatrix}-10^{-6}\\ 1\end{bmatrix} + d\begin{bmatrix}-1\\ 1\end{bmatrix} \qquad (5.26)$$

Setting h = 0.1 and using the initial conditions as the starting vector y^o_n enables a, b, c and d to be computed exactly as

$$a = c = -999999$$

and $$b = d = -1.$$

Now in (5.26)

$$\{1 + h10^6 + (h10^6)^2/2 + \ldots + \rho(h10^6)^6\} = 6.049157816\ e26$$

and

$$\{1 + h + h^2/2 + \ldots + \rho h^6\} = 6.059376119\ e\ 2$$

with ρ = 4.81890304 e -4. The residual vector, r, in (5.26) is then

$$r = \begin{bmatrix} 6.04915781\ e\ 26 \\ -6.04915781\ e\ 32 \end{bmatrix} \qquad (5.27)$$

which is clearly parallel to $[-10^{-6},1]^T$ which corresponds to the dominant eigenvalue $\lambda = -10^6$. This quantity is reproduced by the BRK methods correctly when integrating this problem, hence the residual vector is calculated correctly.

The next stage to check is the back substitution which will find the displacement vector, δ. This requires the solution to

$$[\alpha I + \beta hJ]^6\delta = r \qquad (5.28)$$

by first forming the factorization and then performing six back substitutions in the normal way.

Selecting α and β to maximize the rate of convergence of the Newton process, ie. minimize the amplification factor, results in $(\alpha I+\beta hJ)$ being evaluated as

$$\begin{bmatrix} 1.096622667508114\ e\ 0 & -2.9073996092294181\ e-2 \\ 2.90739960924137\ e\ 4 & 2.90751217896051 20\ e\ 4 \end{bmatrix} \qquad (5.29)$$

with an amplification factor of 2.51e-2.

Thus (5.28) can be solved by using the residual vector, (5.27), by first solving

$$(1.09 \ldots e0)\delta_1 - (2.90 \ldots e-2)\delta_2 = 6.04 \ldots e26$$

$$(2.90 \ldots e4)\delta_1 + (2.90 \ldots e4)\ \delta_2 = -6.04 \ldots e32$$

and using this result as the right-hand side of (5.28) to perform the next back substitution. The displacement vector produced after the first back substitutions is

$$\delta_1 = 2.0805291579\ e22$$

$$\delta_2 = -2.0805291581\ e28$$

which is clearly parallel to the eigenvector $[-10^{-6},1]^T$ which corresponds to the dominant eigenvalue -10^6. Carrying out all six back substitutions lead to the displacement vector set out in Table 5.2.

Clearly after the 6 back substitutions have been performed the displacement vector is not parallel to the dominant eigenvalue, in fact it is parallel to $[-1,1]^T$ which corresponds to the eigenvalue -1. Hence the process has managed to switch from following the dominant eigenvalue to following the other, the mechanism being the same as in the inverse power method, eg. Burden et al.[1978].

The 6-stage 5th order BRK method advances the solution by using

$$y_{n+1} = y_n + h\sum_{i=1}^{6} c_i k_i \qquad (5.30)$$

which can be expressed as

$$y_{n+1} = y_n + h[c_1 J y_{n+1} + \ldots + c_6 J y_{n+1}] \qquad (5.31)$$

Now

$$y_1 = [I - hJ + (hJ)^2/2 - + \ldots + \rho(hJ)^6]y_o \qquad (5.32)$$

and we know

$$y_1^{(o)} = y_o \qquad (5.33)$$

and that

$$y_1^{(o)} + d = y_o \qquad (5.34)$$

where d is the displacement vector. The correct displacement is from (5.32) and using (5.33)

$$[I - hJ + (hJ)^2/2 - + \ldots + \rho(hJ)^6]y_o - y_o$$

$$= F(hJ)y_o \qquad (5.35)$$

where

$$F(hJ) = [- hJ + (hJ)^2/2 - + \ldots + \rho(hJ)^6] \qquad (5.36)$$

Now y_o is constructed from a linear combination of the eigenvectors of

J, (5.25b). Thus (5.35) becomes

$$F(hJ)[cV_1 + dV_2] = cF(hJ)V_1 + dF(hJ)V_2$$
$$= cF(-10^6h)V_1 + dF(-h)V_2$$
$$= c(6.049157816\ e26)V_1 + d(6.05366119\ e2) \qquad (5.37)$$

when h = 0.1 and F(hJ) is obtained from (5.26). The two constants c and d are also computed from (5.26) as c = -999999 and d = -1. Clearly the first term will dominate and thus the displacement vector produced by (5.28) should be parallel to the eigenvalue V_1, ie. $[-10^{-6},-1]^T$. As the process described above does not do this then instead of convergence being attained quickly, it will be an extremely slow process or not obtained at all.

If the denominator of the stability function is of high degree in q then the method will never work satisfactorily. For lower order denominators the method will be more successful but the inverse power method effect will still be present and is likely to reduce the stiffness ratio that can be successfully tackled. Furthermore if a problem with eigenvalues $\lambda = -1, -10^6$ and 1 is considered, this is easily constructed by simply adding in $y' = y_3$, then the approximate factorization will make convergence slow, if obtainable at all.

5.1.3 Computing to extra precision

When the iteration matrix is computed as singular, it is not because the system of equations have become unsolvable but rather due to the inaccuracies in the representation of real numbers on a computer. By working to extra precision the problem can, to some extent, be avoided.

The Prime 550 computer, that has been used for most of the computational work, supports a 32 bit word length as standard.

However, when extra precision is required the double precision arithmetic option can be envoked. This doubles the size of any single variable to 64 bits, this comprises of a 1 bit sign, a 47 bit mantissa and a 16 bit exponent, Figure 5.1.

By employing the REAL*16 option, in Prime FORTRAN 77, the standard word length can be quadrupled to 132 bits per variable. Using this extended precision results in the computations being performed much more accurately and hence the likelihood of the iteration matrix being computed as singular is less.

However, there is a very heavy penalty, in terms of CPU time, to pay for demanding extra precision. Table 5.3 and 5.4 display results for timing round a loop to compute various quantities. Clearly the extra precision increases the CPU times by a factor in excess of 100. The process does, however, alleviate the problem of singular iteration matrices to a certain extent. Table 5.5 compares results for running problem p5.1, with α=6, for a standard 5th order method in double precision and for one in quadruple precision. The low number of Jacobian evaluations, JE, indicate that the extra precision is working, but at a high computational cost.

5.1.4 Decrease order

This singular Jacobian problem is extremely pronounced with high order methods due to the iteration matrix becoming dependent upon high powers of J. Thus one simple solution is to decrease order when the problem is encountered. The code developed here is designed to facilitate the changing of order and hence a simple strategy for decreasing order when the singular problem occurs is permissible. Although this is not the ideal solution, it is felt that there is no simple solution to this

problem for BRK methods. Thus if more than five step failures are due to the iteration matrix then the order is decreased, this continues until first order is reached.

5.2 Incorrectly calculated iteration matrix

Consider integrating

$$\frac{dy_1}{dx} = -0.04y_1 + 0.01y_1y_3$$

$$\frac{dy_2}{dx} = 400y_1 - 100y_2y_3 - 3000y_2^2 \qquad \text{(p5.3)}$$

$$\frac{dy_3}{dx} = 30y_2^2$$

with initial conditions $y(0)=[1,0,0]^T$ by the 2-stage 2nd order BRK methods given by

$$y_{n+1} = y_n + h(k_1 + k_2)/2$$

$$k_1 = f(x_{n+1},\ y_{n+1}) \qquad (5.30)$$

$$k_2 = f(x_{n+1} - h,\ y_{n+1} - hk_1)$$

The Jacobian matrix of p5.3 can clearly be formed and assuming that the problem is slowly varying then the iteration matrix is

$$[I - hJ + (hJ)^2/2] \qquad (5.31)$$

which, taking an initial step of 2.e-4 and using the initial values, can be constructed as approximately

$$\begin{bmatrix} 1.0000008 & 0 & 0 \\ -0.08 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (5.32)$$

For the 2-stage 2nd order BRK method the residual vector can be formed initially as

$$r = y_1 - y_0 - h(k_1 + k_2)/2 \qquad (5.33)$$

where y_0 is given by p5.3 and consequently the initial approximation for y_1 is y_0. The k value required can be computed as

$$hk_1 = [-8.e\text{-}6,\ 8.e\text{-}2,\ 0.e0]^T$$

$$hk_2 = [-8.e\text{-}6,\ 7.6e\text{-}2,\ 3.84e\text{-}5]^T$$

and thus the residual vector is

$$r = [-8.e\text{-}6,\ 7.8e\text{-}2,\ 1.92e\text{-}5]^T$$

which is then used to generate the iteration matrix numerically. This is computed as

$$\begin{bmatrix} 1.0000007987\ e\ 0 & 0.0000000000\ e\ 0 & 5.9604168715\ e\text{-}8 \\ -7.6160321012\ e\text{-}2 & 9.5200013741\ e\text{-}1 & -8.4033373041\ e\text{-}4 \\ -3.8400008634\ e\text{-}5 & 4.7999863720\ e\text{-}4 & 1.0000004033\ e\ 0 \end{bmatrix}$$

which is clearly a good approximation to (5.32). Thus the displacement vector can be computed and the process continued resulting in a converged solution. This whole process works adequately while h is kept small, however, when h is increased by any significant amount the residual vector becomes very large in magnitude and the iteration matrix computed is totally incorrect. This can be simulated by considering an initial step of 0.2, which is not unreasonable especially if a high tolerance is requested.

Now rewriting the k values of (5.30) as

$$k_i^i = f_i(x_{n+1},\ y_{n+1}^1,\ y_{n+1}^2,\ y_{n+1}^3)\quad i = 1,2,3$$

$$k_2^i = f_i(x_{n+1}-h,\ y_{n+1}^1-hk_i^1,\ y_{n+1}^2-hk_i^2,\ y_{n+1}^3-hk_i^3)\quad i = 1,2,3$$

where

$$f_1 = -0.04y_1 + 0.01y_1y_3$$

$$f_2 = 400y_1 - 100y_2y_3 - 3000y_2^2$$

$$f_3 = 30y_2^2$$

At the initial step the approximation for y_1 is y_0 and the k_1 values can be constructed as

$$k_1^1 = f_1(0.2,\ 1.0,\ 0.0,\ 0.0)$$
$$= -0.04$$

$$k_1^2 = f_2(0.2,\ 1.0,\ 0.0,\ 0.0)$$
$$= 400$$

$$k_1^3 = f_3(0.2,\ 1.0,\ 0.0,\ 0.0)$$
$$= 0.0$$

and k_2 as

$$k_2^1 = f_1(0.0,\ 1-0.2(-0.04),\ 0-0.2(400),\ 0-0.2(0))$$
$$= f_1(0.0,\ 1.008,\ -80.0,\ 0.0)$$
$$= -0.04032$$

$$k_2^2 = f_2(0.0,\ 1.008,\ -80.0,\ 0.0)$$
$$= -19199596.8$$

$$k_2^3 = f_3(0.0,\ 1.008,\ -80.0,\ 0.0)$$
$$= 192000$$

which results in a residual vector of

$$r = [8.032\ e\text{-}3,\ 1.91991968\ e6,\ 1.92\ e4]^T$$

The fact that the residual vector is large in magnitude is itself not necessarily a problem. The problem arises when the iteration matrix is computed, it should be approximately

$$\begin{bmatrix} 1.008032 & 0 & 0 \\ -80.32 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The actual iteration matrix computed is, however

$$\begin{bmatrix} 1.00803196\ e\ 0 & -1.1920900\ e\text{-}7 & 8.0000000\ e\text{-}2 \\ 3.83992065\ e\ 6 & -4.8005687\ e\ 4 & -8.0000000\ e\ 2 \\ -3.84000087\ e\ 4 & 4.8006835\ e\ 2 & 1.0000000\ e\ 0 \end{bmatrix}$$

which is clearly incorrect. This iteration matrix results in a displacement vector that forces the solution in completely the wrong direction and usually ends up with an overflow being produced which aborts the program.

There appears to be no simple solution to this problem as it is independent of the order of the methods being employed. The only solution is to check for an exceptionally large residual vector and halve the step size when it occurs. This will ensure that the incorrect matrix is not computed and hence a saving in wasted function evaluations.

This problem, p5.3, is taken from the DETEST package and is problem D2. Consequently all problems in group D of the test package are of a similar nature and BRK methods will perform inefficiently.

No. of Correction terms	Amplification factor
0	2.28e-2
1	1.75e-2
2	8.69e-3
3	5.47e-3
4	1.75e-3

Table 5.1 : Amplification factors for quadratic approximation.

Back subs.	Displacement vector δ_1	δ_2
1	2.0805291579 e22	-2.0805291581 e28
2	7.1556938908 e17	-7.1557094535 e23
3	2.3228628042 e13	-2.4611130639 e19
4	1.2272854297 e12	8.4523974568 e14
5	-1.0909971616 e12	1.0618840290 e12
6	-9.6917511262 e11	9.6917411131 e11

Table 5.2 : Displacement vectors

Quantity computing : sin(x)

No. of times computed : 10000

precision	time (centiseconds)
single real	56
double real	98
quadruple real	11620

Table 5.3 : Timing for computing sin(x)

Quantity computing : a×b

No. of times computed : 100000

precision	time (centiseconds)
single integer	71
double integer	165
single real	70
double real	101
quadruple real	6662

Table 5.4 : Timing for computing a×b

Mode	Log_{10} TOL	FE	JE	Steps	CPU Time
double	-4	129924	332	7484	195.05
	-5	403524	11717	4834	477.86
	-6	531126	16329	5656	623.23
quadruple	-4	1080	34	20	44.56
	-6	2478	68	32	100.71
	-7	8118	213	150	340.35

Table 5.5 : Comparison of double and quadruple precision on problem p5.1.

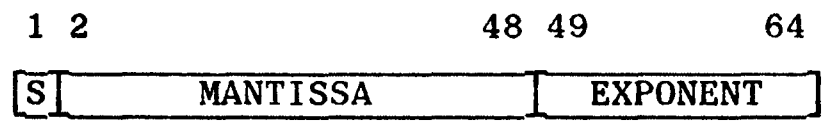


Figure 5.1 : 64 bit word length on a Prime 550

Chapter 6 : TYPE-INSENSITIVE CODE

Many engineering processes give rise to systems of ordinary differential equations which require a fast, accurate and reliable numerical integrator. The user does not want to explore the extensive software available for such calculations. Normally the ODE system used to describe the process may be classified into one of three classes viz. oscillatory, non-stiff or stiff problem, before the integration commences. Each class puts different demands on a numerical solver for efficient integration. However, from studying the initial ODE system there is no simple way of determining the characteristics of the problem, in fact many engineering problems will change characteristics as the integration proceedes. For example, an integrator for a weapon guidance system of a rocket will require a small time step to correctly track the initial stages of the trajectory where the thrust rockets and fins are deployed, if the velocity is to be calculated correctly. The rocket will then go through a relatively steady phase where no vital information can be gathered and hence a large time step should be used. In the final stage the target may be taking evasive action and again a small time step will be required to monitor the flight path accurately, ensuring that the rocket hits the target.

One obvious way of determining the characteristics is to evaluate the eigenvalues of the initial value problem. This, however, may be very time consuming, especially if the system is large or highly non-linear. If the user is unsure of the nature of the system then normally the safest choice is to employ an implicit method, which will clearly be inefficient if the problem is not stiff in any region of the integration range.

This chapter describes an algorithm that takes this important decision of method selection away from the user by monitoring automatically the state of the equations and selecting the most appropriate integrator. The scheme is based on the Backward Runge-Kutta methods of chapter 3, and Explicit Runge-Kutta methods. The complete algorithm will switch from explict solver to implicit solver depending upon the characteristics of the problem at any stage during the integration. A facility for changing order is also incorporated into the code to give additional flexibility.

6.1 Motivation for a type-insensitive code

The motivation for creating a numerical code that is capable of automatically switching between different solvers is readily seen by considering the following system of ODEs,

$$\frac{dy_1}{dt} = -10y_1 + 500y_2$$

$$\frac{dy_2}{dt} = -500y_1 - 10y_2$$

$$\frac{dy_3}{dt} = -4y_3$$

$$t\epsilon[0,64] \qquad \text{(p6.1)}$$

$$\frac{dy_4}{dt} = -y_4$$

$$\frac{dy_5}{dt} = -0.5y_5$$

$$\frac{dy_6}{dt} = -0.1y_6$$

with $y(0) = [1,1,1,1,1,1]^T$

This is a problem based on B5 of the DETEST battery, Enright and Pryce[1983], and is frequently used as a test example. It is

characterised by an initial oscillatory component which is damped out after about 1.5 seconds.

Table 6.1 shows the effect of integrating with an 8th order method in ERK and BRK modes on this problem (p6.1), where Max error is measured as the maximum absolute error over the integration range.

Clearly the CPU time for the ERK method is unacceptable and hence it appears that the explicit method should not be used on this problem at all. However, if the integration range is shortened such that the oscillatory component is not damped out then a very different picture emerges, Table 6.2

The explicit method is far superior over this initial oscillatory stage of the integration. When the oscillations have been damped out the BRK requires only another 24 steps to complete the integration, whereas the ERK method must continue to use a small step.

Clearly the ideal situation, on this problem, would be to integrate up to about 1.5 using an explicit method and then switch over to an implicit method where the step will be allowed to rise sharply. From this example the need for a code that automatically switches numerical solvers is apparent, such codes are often referred to as type-insensitive.

In Chapter 2 three Runge-Kutta methods were generated from one set of basic coefficients viz., ERK, BRK and MRK methods. Chapter 3 showed that BRK methods are often capable of integrating stiff systems of ODEs efficiently, whereas ERK methods are commonly used for the solution of

non-stiff ODEs. Although MRK methods can be made precisely A-stable, it was shown in chapter 2, that they do not perform significiantly better than ERK methods at reproducing the correct characteristics of an oscillatory solution, ie. they cannot correctly predict the frequency of the component unless the step size is severely restricted. We conclude that the oscillatory problems can be integrated efficiently by an explicit method. A schematic overview of the explicit and implicit methods employed in the algorithm is presented in Figure 6.1. The arrows indicate the direction of switching that is used. A flowchart illustrating the switching available at any step of the algorithm is shown in Figure 6.2 for the 3rd order ERK method, the same basic strategy applies to BRK methods.

6.2 Switching integrator

The two classes of integrators employed are the expicit, ERK methods and the implicit, BRK methods. The ideas behind the switching strategies are basically the same regardless of the order of the method currently being employed.

6.2.1 Switching from explicit method to implicit method

The algorithm starts the integration with the explicit method using a very small step, as this is computationally the cheapest and the state of the equations are undetermined initially. The explicit method will be efficient providing that the system remains non-stiff or oscillatory. If the equations are, or become, stiff a change must be made to the stiff solver, hence some means of detecting stiffness is needed. The approach taken here is to use a stiffness detection scheme based on an idea by Shampine and Hiebert[1977].

Assuming that the integration is being performed by an s-stage method of order p+1, (s,p+1), and the error controlled by an embedded (s,p) method then after a successful step the error control mechanism will select a new step size. This step will be the largest possible to meet the accuracy requirements, this may correspond to a q_i value being on the boundary of the methods stability region in which case the step size is restricted for stability reasons. Alternatively all q_i values may be well within the stability region and the step size is restricted by accuracy.

The main requirement, apart from reliability, of the stiffness detection scheme is cheapness. As the scheme is to be employed after every successful step the cost of it must be minimal.

Consider taking a step with the (6,5) and (6,4) Fehlberg methods ie.

$$y_{n+1} = y_n + \sum_{i=1}^{6} c_i k_i$$

$$k_i = f(x_n + ha_i,\ y_n + h\sum_{j=1}^{i-1} b_{ij}k_j) \qquad (6.1)$$

where the constants are given in Table 6.3. At each step six function evaluations at discrete points between x_n and x_{n+1} are available. From these a (6,2) and a (6,1) method can be constructed ie. 2nd and 1st order methods with 6 stages. This is clearly inefficient in a normal explicit scheme, but in this case the function evaluations, k values, are already available from the (6,5) method. Thus a linear combination of them can be taken to form the lower order methods.

For the (6,2) method this will be,

$$\tilde{y}_{n+1} = y_n + \sum_{i=1}^{6} \tilde{c}_i k_i \qquad (6.2)$$

and for the (6,1) method

$$\tilde{y}^*_{n+1} = y_n + \sum_{i=1}^{6} \tilde{c}^*_i k_i \qquad (6.3)$$

where the weights are taken together with the constants from Table 6.3 to generate the required order method.

The two equations to be satisfied for a 6-stage 2nd order method are

$$\sum_{i=1}^{6} \tilde{c}_i = 1$$

$$\text{and} \quad \sum_{i=1}^{6} \tilde{c}_i a_i = 1/2 \qquad (6.4)$$

By incorporating the coefficients from the main (6,5) method the six free parameters can be reduced to four with

$$\tilde{c}_2 = (\tfrac{1}{2} - \sum_{i=3}^{6} \tilde{c}_i a_i)/a_i$$

$$\text{and} \quad \tilde{c}_1 = 1 - \sum_{i=2}^{6} \tilde{c}_i \qquad (6.5)$$

Similarly for the 1st order method, one free parameter can be removed by setting

$$\tilde{c}^*_1 = 1 - \sum_{i=2}^{6} \tilde{c}^*_i \qquad (6.6)$$

The only requirement of this lower order pair is that their stability regions are uniformly larger than that of the (6,5) method.

A computer search was made for a (6,2) and a (6,1) pair which had a stability region uniformly larger than the (6,5) method. The coefficients produced are the same as those published by Shampine and Hiebert[1977] and are given in Table 6.3. On all subsequent plots the

stability regions for each method will be denoted by the following key;

i) MAIN for the main integrator,

ii) ERROR for the integrator used for error control (ERK only),

iii) ORDER for an integrator use for switching order, and

iv) METHOD for an integrator used to switch numerical method.

The absolute stability regions of the 5th order related methods are shown in Figure 6.3. Clearly the embedded explicit methods, ERK 2 and ERK 1, used for the switching of numerical methods are uniformly larger than the main explicit methods.

Similarly for a (3,3) method with a (3,2) method for error control, embedded (3,2) and (3,1) methods for stiffness detection can be found. Table 6.4 shows these coefficients and their stability region are shown in Figure 6.4.

Clearly this idea of embedding a 2,1 pair within the higher order methods for the detection of stiffness, can be extended to higher orders. It cannot, however, be used in conjunction with any ERK pair lower than 3rd order. For this scheme to work a 2nd and 1st order embedded pair are required with stability regions uniformly larger than the main method. Hence if the main method is 2nd order, in 2 stages, then the embedded 2nd order method must have exactly the same stability region as the main method. Thus a switch from ERK 2 to BRK 2 is not provided. The 2nd order coefficients are, however, given for completeness in Table 6.5.

The main method will select a step size that just meets the accuracy requirements. If the step size is restricted on the grounds of accuracy

rather than stability then the lower order embedded pair are unlikely to satisfy the accuracy requirements. We know that the lower order pair are failing for accuracy reasons, and not for stability reasons as its stability region is larger than that of the main method. Thus the system is not stiff at the current integration point.

If the lower order embedded method is repeatedly able to meet the same accuracy requirements then the step size of the higher order method must be restricted for stability reasons. This implies that the problem is becoming stiff. If the embedded method is successful for 50% of the time over 50 consecutive steps then the problem is deemed stiff and a switch to the implicit method advocated.

To increase efficiency the step size is increased by a factor of 5 when the switch is activated. This is primarily due to the "dead band" introduced in the step control algorithm, chapter 4, to save on iteration matrix updates. If the BRK method fails to meet the accuracy requirements with this increased step then a switch back to the explicit method is performed, otherwise the integration continues with the implicit method.

Since the ERK and BRK methods share the same coefficients when a switch is performed, no new coefficients have to be calculated and further parts of the code can still be used. This makes the overheads in switching minimal.

6.2.2 Switching from implicit method to explicit method

In order to ensure that the code is competitive the implicit to explicit switching must also be cheap.

One approach, Shampine[1981], Norsett and Thomsen[1986], is to use the approximation to the Jacobian matrix of the problem, or the exact Jacobian matrix if this is supplied to the code, to form some estimate of the Lipschitz constant. However, in the case of BRK methods this matrix is not computed As described in chapter 3, it is an approximation to the iteration matrix that is computed.

The approach taken here is to generate some non-stiff detection schemes based on the function evaluations available from the implicit method. These will be the k values after the implicit equations have been solved to an acceptable tolerance by the quasi-Newton process. Thus the detection scheme will be a numerical method in its own right.

The only requirements of this new method is that it uses the k values from the implicit method and must have characteristics like an explicit method. In particular it must possess a finite stability region. Taking a linear combination of the k values produced by an implicit BRK method clearly cannot result in a method with a finite stability region. If this where so then the corresponding ERK method would have an infinite stability region!

The general s-stage BRK method is

$$y_{n+1} = y_n + h\sum_{i=1}^{s} c_i k_i \qquad (6.7a)$$

$$k_i = f(x_{n+1} - hb_i,\ y_{n+1} - h\sum_{j=1}^{i-1} a_{ij} k_j) \qquad (6.7b)$$

After a successful step with any Runge-Kutta method the k values are discarded and a new collection generated on the next step. In fact the only value carried forward is the solution y_{n+1}, which becomes y_n

on the next step. By using previous solution values, (y values), the resulting method becomes akin to linear multistep methods and will inherit their deficiencies.

If, however, the k values computed on a step are stored and used on the next step then a new method, an Extended BRK method (EBRK), is generated, which can possess a finite stability region. Thus instead of (6.7a)

$$y_{n+1} = y_n + h\sum_{i=1}^{s} c_i k_i + h\sum_{i=1}^{t} c_i^* k_i^* \qquad (6.8)$$

is used where k_i^* are selected from the k_i, i=1(1)s values from the previous step. Thus the k values on step n are stored and used on step n+1. However, to store every k value is unnecessary as they can be stacked up in one vector viz.

$$\sum_{i=1}^{t} c_i^* k_i^* \qquad (6.9)$$

is calculated, constructed and stored on the nth step and used on the (n+1)th step. Clearly this technique must be employed on the second h step of the Richardson process.

The solution produced by this EBRK will only be reasonable if $q_i = h\lambda_i$, i = 1(1)N is within the stability region of the method for all i such that $Re(q_i) < 0$. The solution of the EBRK method is compared with that obtained by the BRK with the second h step and is regarded as successful if it satisfies the same error tolerance as is employed for the BRK method. However, the object of using the EBRK methods is to determine whether the ERK method could be used to perform the integration efficiently and not to produce a usable solution itself. If the solutions agree for five consecutive steps then q_i is deemed to

be "small" and a switch to the cheaper explicit method performed. The new step size taken by the ERK method is that last used by the BRK method.

The weights for the EBRK method were found numerically by a computer search. Clearly the weights are constrained by the order equations that must be satisfied for a particular order, ie. for a 3rd order method there are four equations to satisfy, Table 1.1. The successful weights being those that would produce stability regions that were finite and matched the stability regions of the corresponding ERK method. Initial testing was performed by trying to find a 3rd order EBRK, for the 3rd order BRK method, with only 1 past k value, ie. t = 1. This, however, did not produce an acceptable stability region. With t set at 2 the EBRK method with stability region shown in Figure 6.5 is produced. This is compared with the stability region of the 3rd order ERK method. The weights for this are

$$c_1 = 1.0100 \qquad c_1^* = 4.2167$$

$$c_2 = -2.0320 \qquad c_2^* = -1.5180$$

$$c_3 = -0.6767$$

The method employed was to guess one coefficient, determine the others by solving four linear equations and plot the corresponding stability region. Remarkably the objective was to make the region as small as possible!

For the 5th order methods a 4th order EBRK was initially tested to determine if the same idea would work for higher orders. Suitable weights found were, using t = 4,

$$c_1 = 0.00000 \qquad c_1^* = 0.45371$$

$$c_2 = -0.91475 \qquad c_2^* = 0.07037$$

$$c_3 = 3.37465 \qquad c_3^* = 0.16789$$

$$c_4 = 0.00000 \qquad c_4^* = -0.05868$$

$$c_5 = -0.29977$$

$$c_6 = -1.79342$$

Initial testing was performed with this 4th order EBRK. The method proved very reliable and was always able to detect when the problem became non-stiff. Ideally a 5th order EBRK method should be used. However, no suitable method was found and since the 4th order method proved so effective it was retained. Simple numerical testing with a 2nd order EBRK for the use with the 3rd order BRK demonstrated that the order of the EBRK need not match exactly the order of the BRK method. However, the 3rd order EBRK method was retained for use with the 3rd order BRK method.

Similarly a EBRK for the 2nd order BRK method is required. This can be generated by the use of one past value. One suitable choice of weights is

$$c_1 = 0.5 \qquad c_1^* = 1.0$$

$$c_2 = -0.5$$

and using the coefficients of Table 6.5 to produce the stability regions in Figure 6.6. This method is 2nd order.

6.3 Switching order

To meet the requirements of the user and to make the code more flexible it should ideally be able to select the appropriate order. The orders employed for this code are 2, 3 and 5 for the explicit methods and orders 1, 2, 3 and 5 for the implicit ones. The main orders in the

code are 3 and 5 with the others playing a supporting role in case of serious failures. The 2nd order ERK method being employed mainly so that the algorithm can recover from the use of lower order BRK methods eg. being forced to low order if the iteration matrix is regularly computed as singular.

6.3.1 Order Reduction

As the decision to switch order must be inexpensive, the k values produced by the main integrator must be used. If the integrator has order p then clearly, a method of order less than p can be embedded within this method such that the same function evaluations are used.

When either the ERK or BRK 5th order method is being used a decision must be made as to whether to switch down to the 3rd order method. By embedding a 3rd order method within the 6-stages of the 5th the test can be performed. For the ERK method an embedded 3,2 pair will be required for the error estimation while for the BRK method the same 3rd order method used in backward mode can be employed. The weights were determined numerically by the same method as described in section 6.2.2. The final values chosen were

	3rd order	2nd order
c_1 =	0.0831292	0.0031290
c_2 =	-0.0029698	0.0070320
c_3 =	0.6187300	0.1285200
c_4 =	0.1847500	0.1234860
c_5 =	0.0800000	-0.0656610
c_6 =	0.0363636	0.8034940

Their stability regions, used in conjuncation with the coefficients of Table 6.3 are shown in Figures 6.7 and 6.8 for ERK and BRK modes

respectively.

The embedding error test, for the ERK method, is performed by taking the difference between the 3rd order method and the 2nd after the 5th order ERK method has taken a successful step. A Fehlberg type test is then carried out, described in chapter 3, to determine if the lower order method can match the required accuracy.

For the BRK method the solution obtained from the lower order method, over the second h step, is compared with the BRK solution produced there.

The same embedding process can be used for a 2nd order, plus 1st for the ERK method, within the 3rd order methods. The weights in this case are

	2nd order	1st order
c_1 =	1/3	1/5
c_2 =	1/3	1/5
c_3 =	1/3	3/5

and the same error control test as described above is used. Figures 6.9 and 6.10 present the stability region of these methods used in conjunction with the coefficients of Table 6.4.

6.3.2 Increasing order

All of the switching strategies developed so far rely on information readily available at the end of a successful step. The only extra expense involved is a few multiplications and additions. Various ways of increasing order were considered that involved computing extra function evaluations. Only two possibilities came to light that would

not require any additional work. The first was to use another extended method, ie. using previous k values. However, it proved difficult to generate the EBRK methods with the required stability and accuracy properties. When employing the 3rd order method the proposed change is to 5th order, hence the extended method will require at least 6-stages. Even by taking all the k values from the previous step a 5th order method with the correct stability region is not guaranteed. The EBRK 4th order method required 4 previous values, 10-stages in total to produce the required stability region.

The much simpler option is to continue with the current order method until the step fails for accuracy reasons. At that point an order increase is advocated. This method was employed for both the ERK and BRK methods. If the ERK method is being used, than a step size failure must be due to accuracy if the stiffness detection scheme was not triggered. Thus the stiffness prediction scheme must be given priority over order change. This very simple order increase scheme is justified purely on the grounds that numerical tests indicate that it works reasonably well as shown in section 6.5

6.4 General comments

Although two separate numerical methods, ERK and BRK, are employed in the complete algorithm they are closely related. This makes the final code relatively compact as often the same piece of code can be used for both methods by simply switching some of the parameters. For example one subroutine GFUN calculates the required k values; for the ERK method this is called as

CALL GFUN(X, Y_n, Y_{n+1}, H,. . .)

and can be called in conjunction with the BRK as

CALL GFUN(X, Y_{n+1}, Y_n, -H,. . .)

In all, five subroutines can be used for both explicit and implicit methods in this way.

6.5 Numerical results

The numerical results will be split into sections to test the correct behaviour of all the switching strategies described earlier. The headings for each table are as used in the previous chapters with significant figures accuracy measured at the end of the integration range over the relevant component, ie. y_3, y_3, y_4 and y_1 for problems p3.1, p3.2, p3.3 and p3.4 respectively.

6.5.1 Integrating non-stiff problems

In this section the ability of the code to select methods for non-stiff problems is tested. Two problems are considered, one is problem p3.1 with the coefficient -10^6 of y_1 changed to -1, yielding problem p3.1a, to make the problem non-stiff. The other is problem p3.2 with a change in the initial conditions of y_2 and y_3 to -1, producing problem p3.2a. Clearly in this state stiffness should not be detected and thus the explicit method should be used throughout the integration range. The results of integrating these two problems with ERK3, ERK5, (explicit methods of order 3 and 5), BRK3, BRK5 and SARK are shown in Table 6.6 and 6.7.

The most efficient method for p3.1a is clearly the high order explicit method with the implicit methods being inefficient. Clearly SARK chooses the correct method of integration by staying with the high order explicit method. The CPU times produced by SARK are higher than the ERK5 because of the slight overheads incurred in the switching

stratagies.

For problem p3.2a a similar pattern emerges with the explicit methods being far superior to the implicit ones. Again SARK can correctly deduce that the problem can be solved more efficiently by an explicit method and thus no switching occurs. (Table 6.7).

The other occasion when a non-stiff integrator is required is when a stiff problem becomes non-stiff. This can be simulated by commencing the integrating of a non-stiff problem with a stiff integrator. The detection scheme should detect that the problem is non-stiff and switch accordingly. Table 6.8 and 6.9 display results that compare BRK5 with SARK, run in this manner, on problems p3.1a and p3.2a. By commencing the integration of SARK with the 5th order implicit method it clearly switches over to the cheaper explicit method when the problem is deemed to be non-stiff. As a result the CPU times produced are superior to those produced by the 5th order implicit method. For p3.1a the ERK methods integrate for 90.4% of the integration range at a tolerance of 1.e-3 and for 56.1% at a tolerance of 1.e-6. On problem p3.2a these percentages are even higher at 90.5% (1.e-3) and 81.8% (1.e-6), clearly showing that lack of stiffness is detected.

6.5.2 Integrating a stiff problem

This section examines the ability of the code to select the correct method when confronted with a stiff problem. The switching of order is also tested. Clearly no explicit method is able to integrate a stiff problem efficiently and hence SARK must switch over to an implicit method. The results of running the stiff problems p3.1 and p3.4 are shown in Tables 6.10 and 6.11. For problem p3.1 the low order implicit

method is the most efficient, hence SARK should switch over to this method, which it does, producing comparable accuracy in a faster time. Problem p3.4 on the other hand is integrated quicker by the high order method at a tight tolerance and by the lower order method at a high tolerance. SARK manages to switch to the appropriate method and produce superior results to either of the implicit methods at both tolerances. For both problems the BRK methods were used for 99.9% of the integration. These results show that, not only is the switching of numerical methods handled correctly, but also the most appropriate order is selected.

6.5.3 General results

Three problems are considered in depth and compared with results from both Gear's and Adams methods. The first problem is the one considered at the start of this chapter as a motivation for a type-insensitive code. Also considered is the van der Pol equation broken down into a system of first order differential equations

$$\frac{dy_1}{dt} = y_2$$

$$\frac{dy_2}{dt} = -y_1 + \lambda(1 - y_1^2)y_2 \qquad t\epsilon[0,a] \qquad (p6.2)$$

with initial conditions $y(0) = [1,1]^T$. This problem is known to change its characteristics during the integration range and is often cited in literature, Petzold[1983], Norsett and Thomsen[1986], as a test problem for type-insensitive codes. By selecting different values of λ and the endpoint of the integration range, a, the problem possesses different characteristics. Three values of λ, 5, 10 and 100, are considered over two different integration ranges a = 10 and 100. With λ = 5 and a = 10 the problem is deemed non-stiff. As λ and a are increased the

problem becomes stiffer. At the two extremes the problem is clearly defined as either stiff or non-stiff and hence a general purpose integrator may be unable to be as competitive as one specially designed for a specific class of problems. Finially problem p6.3

$$\frac{dy_1}{dt} = y_2$$

$$\frac{dy_2}{dt} = -(100^2 + 1)y_1 - 2y_2 \qquad t\epsilon[0,10] \quad \text{(p6.3)}$$

with initial conditions $y(0) = [1,1]^T$ is used to test the ability of SARK to cope with an oscillatory solution. The results of integrating these three problems are set out on in Tables 6.12 to 6.19. Each set of results is produced by modifing the DETEST package, and using the normalised statistics produced. This takes into account the different way that codes control the local error. The quantity $\log_{10}$ accuracy is the expected accuracy of the numerical method over the integration range for all components. Further details about this, the DETEST package and the statistics produced are described in the next chapter.

Problem p6.1 was integrated with a small initial step, 1.e-6, to ensure that the BRK methods did not damp out the initial oscillatory solution. Both GEAR and ADAMS performed badly, Table 6.12 and Figure 6.11, with the CPU times being over 200 times worse than that of SARK. The low number of Jacobian evaluations for SARK indicates that the code is integrating over the oscillatory phase with the explicit methods and then switching to the implicit method when the solution smoothes out.

The results for various parameters λ and α for problem p6.2 are shown in Tables 6.13 - 6.18 and graphically in Figures 6.12 - 6.17. When λ = 5 and α = 10, Table 6.13 and Figure 6.12, the problem is non-stiff and

clearly the ERK methods are more efficient than the BRK methods. Surprisingly ADAMS performs as badly as GEAR and both are inferior to SARK which correctly chooses the high order explicit method.

Increasing the value of λ to 10 introduces more stiffness into the system. This is apparent from Table 6.14 and Figure 6.13 which demonstrates that GEAR is superior to the ADAMS, but it is still slower than SARK.

With λ = 100, Table 6.15, ADAMS is very inefficient as shown by Figure 6.14. Although GEAR is slightly more efficient than SARK, SARK does select the most efficient method available. It is only the high number of function evaluations required for all Runge-Kutta based methods that makes SARK slightly less efficient. If the cost of performing the function evaluations are removed from the overall CPU time then the overheads of the method can be assessed. The DETEST package provides statistics for these figures. As both SARK and GEAR are using a numerical Jacobian this is valid for both methods. When this is done SARK far outperforms GEAR, the column entitled OVHD of Table 6.15. The fact that the efficiencies of Gear and Adams are so different indicate that the problem is stiff and as a result a specialized stiff integrator must be more efficient than a type-insensitive code.

Resetting λ to 5 and allowing a to increase to 100 returns the problem to the non-stiff state as indicated by the results of Table 6.16, Figure 6.15. Again SARK correctly selects the most efficient method and is far superior to either of the multistep codes.

Increasing λ to 10 produces a stiff problem when the tolerance is high

and a non-stiff problem when it is low, Table 6.17, Figure 6.16. As the tolerance is decreased the code recognises that the problem can be integrated more efficiently by the explicit method and thus stays with ERK5. This problem also indicates that stiffness is not just dependent upon the problem being solved but is also dependent upon the accuracy requested.

With λ = 100 and a = 100, Table 6.18 and Figure 6.17, the problem possesses a severe spike in the second component during the integration. For most of the integration range y_2 is approximately -0.1, but at one particular point it shoots down to -133.7 and back to -0.1 almost immediately. Norsett and Thomsen[1986] use their switching code to locate the exact position of this spike. They claim that the spike is at 81.92. This is incorrect. The spike is in fact located at 81.2 which can be picked up accurately by SARK. This can be confirmed by integrating the problem with a very small time step and tight tolerance by GEAR. For this choice of parameters ADAMS method is very inefficient, again indicating that the problem is stiff. Overall GEAR is marginally more efficient than SARK in terms of CPU time for a requested accuracy. Clearly, however, the cost in overheads of GEAR is much higher than the associated cost with SARK even though SARK changes integrator as well as order. The cost of the overheads in SARK is between 55% and 70% of the total CPU time, (this depends upon the integrator being employed), whereas about 93% of the time for GEAR is involved with overheads.

The final problem considered is p6.3. This is characterised by an oscillatory solution and thus the explicit method will be superior. This is confirmed by Table 6.19 and Figure 6.18. SARK correctly

deduces that the step is being restricted for accuracy reasons and thus the explicit method is used throughout.

These three problems demonstrate that SARK is capable of correctly selecting the most suitable method available to it and that it is often superior to specialized integrators. Clearly when the characteristics of the problem are such that it can be easily categorized as stiff or non-stiff, then a specialized integrator may be superior. However, it is often difficult to categorize the problem and furthermore many users of codes are not interested in attempting to categorise the problem. In such cases SARK is to be recommended.

	Min. step	Max. step	Max. error	Error at x_{end}	No. of steps	CPU time
BRK	2.3e-3	3.46e0	3.2e-6	6.7e-9	536	87.7
ERK	1.3e-3	1.1e-2	1.5e-4	3.0e-6	1127	497.0

Table 6.1 : Integrating upto $x_{end} = 64.0$

	Min. step	Max. step	Max. error	Error at x_{end}	No. of steps	CPU time
BRK	2.3e-3	1.3e-2	3.2e-4	2.3e-7	512	78.9
ERK	1.3e-3	1.1e-2	1.6e-3	1.3e-5	298	25.8

Table 6.2 Integrating upto $x_{end} = 1.57$

0	0						
$\frac{1}{4}$	$\frac{1}{4}$						
$\frac{3}{8}$	$\frac{3}{32}$	$\frac{9}{32}$					
$\frac{12}{13}$	$\frac{1932}{2197}$	$\frac{-7200}{2197}$	$\frac{7296}{2197}$				
1	$\frac{439}{216}$	-8	$\frac{3680}{513}$	$\frac{-845}{4104}$			
$\frac{1}{2}$	$\frac{-8}{27}$	2	$\frac{-3544}{2565}$	$\frac{1859}{4104}$	$\frac{-11}{40}$		
	$\frac{16}{135}$	0	$\frac{6656}{12825}$	$\frac{28561}{56430}$	$\frac{-9}{50}$	$\frac{2}{55}$	(6,5)
	$\frac{25}{216}$	0	$\frac{1408}{2565}$	$\frac{2197}{4104}$	$\frac{-1}{5}$	0	(6,4)
	.139682	-.198633	.724462	.428953	-.141485	.047041	(6,2)
	.084227	-.163140	.761013	.405846	-.131970	.044024	(6,1)

Table 6.3a : Coefficients for 5th order related methods

(6,5) main method
(6,4) error control for ERK method
(6,2), (6,1) } stiffness detection

Table 6.3b : Description of methods of Table 6.3a

0				
$\frac{1}{2}$	$\frac{1}{2}$			
1	-1	2		
	$\frac{1}{6}$	$\frac{2}{3}$	$\frac{1}{6}$	(3,3) main method
	$\frac{2}{10}$	$\frac{6}{10}$	$\frac{2}{10}$	(3,2) error control
	$\frac{3}{25}$	$\frac{19}{25}$	$\frac{3}{25}$	(3,2) } stiffness detection
	$\frac{1}{100}$	$\frac{84}{100}$	$\frac{15}{100}$	(3,1) }

Table 6.4 : Coefficients for 3rd order related methods

0	0	
1	1 0	
	½ ½	(2,2) main method
	1 0	(2,1) error control for ERK method

Table 6.5 : Coefficients for 2nd order related methods

Method	Order	Log_{10} TOL	FE	JE	Steps	Sig. Figs. Accuracy	CPU Time
ERK	3	-3	108	-	36	4.16	0.21
		-6	849	-	283	7.08	1.07
ERK	5	-3	126	-	21	4.85	0.17
		-6	354	-	59	8.31	0.40
BRK	3	-3	468	14	68	4.37	0.63
		-6	1374	10	282	6.61	1.98
BRK	5	-3	1566	14	106	3.71	1.73
		-6	1704	13	90	7.09	1.74
SARK		-3	126	0	21	4.85	0.20
		-6	354	0	59	8.31	0.54

Table 6.6 : Results for non-stiff problem p3.1a

Method	Order	Log_{10} TOL	FE	JE	Steps	Sig. Figs. Accuracy	CPU Time
ERK	3	-6	429	-	143	2.61	0.70
		-9	4101	-	1367	5.58	6.77
ERK	5	-6	126	-	21	1.79	0.20
		-9	366	-	61	4.51	0.53
BRK	3	-6	843	13	90	1.21	1.23
		-9	2937	13	414	4.52	4.70
BRK	5	-6	1308	19	68	3.16	1.80
		-9	1980	15	122	5.33	2.66
SARK		-6	126	0	21	1.79	0.21
		-9	366	0	61	4.51	0.55

Table 6.7 : Results for non-stiff problem p3.2a

Method	Order	Log_{10} TOL	FE	JE	Steps	Sig. Figs. Accuracy	CPU Time	% ERK
BRK	5	-3	1566	14	106	3.71	1.73	-
		-6	1704	13	90	7.09	1.74	-
SARK		-3	702	14	42	7.60	1.08	90.4
		-6	1290	13	71	9.59	1.69	56.1

Table 6.8 : Comparison of BRK5 and SARK on problem p3.1a

Method	Order	Log_{10} TOL	FE	JE	Steps	Sig. Figs. Accuracy	CPU Time	% ERK
BRK	5	-3	1260	25	44	0.72	1.72	-
		-6	1308	19	68	3.16	1.80	-
SARK		-3	576	13	31	1.14	0.99	90.5
		-6	744	10	53	3.22	1.26	81.8

Table 6.9 : Comparison of BRK5 and SARK on problem p3.2a

Method	Order	Log_{10} TOL	FE	JE	Steps	Sig. Figs. Accuracy	CPU Time	% BRK
BRK	3	-3	516	12	58	3.18	0.75	
		-6	1950	27	346	6.45	2.96	
BRK	5	-3	2136	36	132	3.77	2.29	
		-6	2958	36	222	4.57	3.38	
SARK		-3	585	12	113	3.04	1.50	99.9
		-6	1521	12	301	6.43	2.66	99.9

Table 6.10 : Results for stiff problem p3.1

Method	Order	Log_{10} TOL	FE	JE	Steps	Sig. Figs. Accuracy	CPU Time	% BRK
BRK	3	-4	6153	33	1288	1.31	7.68	
		-7	31329	39	6850	3.25	69.19	
BRK	5	-4	50496	87	2690	1.68	43.60	
		-7	67158	63	2828	4.26	54.93	
SARK		-4	1545	14	245	1.47	1.82	99.9
		-7	11316	45	1691	3.90	11.35	99.9

Table 6.11 : Results for stiff problem p3.4

Method & Order	Log_{10} Accuracy	FE	JE	Steps	CPU Time
ERK 3	-3	53549	-	17848	86.98
	-4	56437	-	18811	91.30
	-5	62790	-	20929	100.81
	-6	76488	-	25495	122.87
	-7	105774	-	35257	170.13
	-8	168464	-	56153	269.70
ERK 5	-4	56420	-	9402	83.00
	-5	57741	-	9622	84.30
	-6	59865	-	9976	87.18
	-7	63221	-	10535	91.69
	-8	68462	-	11409	99.42
BRK 3	-5	14532	23	1439	26.64
	-6	39558	23	3987	76.87
	-7	68964	53	7504	143.22
BRK 5	-3	6949	17	341	10.85
	-4	10999	20	557	17.20
	-5	14273	23	729	22.37
	-6	20764	25	1085	32.84
	-7	28408	25	1509	45.27
	-8	36179	23	1944	58.04
	-9	43873	20	2381	70.83
SARK	-6	4078	8	583	6.78
	-7	8178	8	1248	13.91
	-8	17942	7	2859	30.89
GEAR	-2	58815	2274	44933	663.13
	-3	58848	2272	44941	668.04
	-4	58074	2245	44446	662.07
	-5	54499	2113	41685	630.86
	-6	63473	2458	48562	713.57
ADAMS	-2	97452	-	42328	583.47
	-3	128331	-	64865	969.94
	-4	132494	-	70030	1022.82
	-5	106536	-	68430	720.44
	-6	159207	-	98408	1404.96
	-7	90477	-	45774	522.39

Table 6.12 : Problem p6.1 (Figure 6.11)

Method & Order	Log_{10} Accuracy	FE	JE	Steps	CPU Time
ERK 3	-4	535	-	166	0.48
	-5	1070	-	351	0.97
	-6	2270	-	753	2.11
	-7	4872	-	1621	4.42
	-8	10501	-	3496	9.49
ERK 5	-4	299	-	41	0.17
	-5	418	-	62	0.27
	-6	634	-	97	0.43
	-7	953	-	152	0.68
	-8	1493	-	244	1.04
BRK 3	-3	1672	49	45	0.93
	-4	2453	33	101	1.49
	-5	3986	45	244	2.73
	-6	8233	52	691	6.42
BRK 5	-3	5228	47	123	2.55
	-4	5349	33	130	2.56
	-5	5083	25	125	3.43
	-6	6209	9	188	3.16
	-7	7842	9	256	4.08
SARK	-4	301	0	42	0.27
	-5	420	0	63	0.37
	-6	636	0	98	0.57
	-7	954	0	153	0.85
	-8	1492	0	243	1.33
GEAR	-2	327	29	150	1.33
	-3	389	31	194	1.66
	-4	459	32	270	2.16
	-5	558	38	353	2.66
	-6	722	48	505	3.48
ADAMS	-2	370	-	224	1.43
	-3	450	-	306	1.97
	-4	447	-	309	2.05
	-5	551	-	413	2.76
	-6	640	-	480	3.35

Table 6.13 : Problem p6.2, λ=5, a=10 (Figure 6.12)

Method & Order	Log_{10} Accuracy	FE	JE	Steps	CPU Time
ERK 3	-5	796	-	246	0.70
	-6	1694	-	546	1.54
	-7	3957	-	1313	3.66
	-8	9496	-	3162	8.65
ERK 5	-4	451	-	67	0.28
	-5	588	-	87	0.37
	-6	822	-	122	0.54
	-7	1168	-	187	0.76
	-8	1756	-	289	1.15
BRK 3	-4	1679	39	55	0.98
	-5	3111	30	176	2.09
	-6	6009	38	419	4.31
BRK 5	-4	8099	26	220	4.09
	-5	5905	30	150	2.89
	-6	6447	32	151	3.10
	-7	8368	43	196	4.04
SARK	-4	459	0	68	0.41
	-5	592	0	88	0.52
	-6	826	0	123	0.72
	-7	1172	0	187	1.05
	-8	1756	0	289	1.63
GEAR	-3	316	28	151	1.29
	-4	410	32	212	1.79
	-5	511	34	302	2.37
	-6	605	40	387	2.97
ADAMS	-3	547	-	339	2.31
	-4	615	-	379	2.68
	-5	729	-	528	3.77
	-6	762	-	514	3.81

Table 6.14 : Problem p3.2, λ=10, a=10 (Figure 6.13)

Method & Order	Log_{10} Accuracy	FE	JE	Steps	CPU Time	OVHD
ERK 3	-4	4170	-	1078	3.39	
	-5	4165	-	1080	3.47	
	-6	4210	-	1086	3.52	
	-7	4277	-	1111	3.57	
	-8	4434	-	1164	3.72	
ERK 5	-5	4691	-	779	3.48	
	-6	4707	-	782	3.16	
	-7	4724	-	785	3.10	
	-8	4755	-	790	3.13	
	-9	4808	-	797	3.16	
BRK 3	-4	440	21	7	0.22	
	-5	346	16	9	0.19	
	-6	485	18	19	0.29	
	-7	686	15	43	0.48	
BRK 5	-4	8120	42	177	3.81	
	-5	5838	52	126	2.84	
	-6	4818	59	86	2.14	
SARK	-4	1429	32	57	0.81	0.36
	-5	1132	25	46	0.59	0.23
	-6	1051	23	45	0.55	0.22
	-7	1595	16	85	0.95	0.41
GEAR	-5	69	10	28	0.24	0.21
	-6	127	16	54	0.45	0.41
	-7	187	19	96	0.73	0.67
ADAMS	-4	45772	-	32220	204.58	
	-5	43673	-	33582	208.95	

Table 6.15 : Problem p6.2, λ=100, a=10 (Figure 6.14)

Method & Order	Log_{10} Accuracy	FE	JE	Steps	CPU Time
ERK 3	-3	5503	-	1661	5.61
	-4	10829	-	3522	11.31
	-5	23486	-	7795	24.84
	-6	51847	-	17262	54.63
	-7	114811	-	38247	120.85
ERK 5	-3	3919	-	547	2.78
	-4	5466	-	801	4.04
	-5	7643	-	1181	5.70
	-6	10874	-	1751	8.09
	-7	16176	-	2627	12.30
	-8	24323	-	4000	18.46
BRK 3	-3	18293	486	470	11.49
	-4	37723	411	1583	26.54
	-5	87198	942	5189	68.07
BRK 5	-3	57424	484	1131	29.90
	-4	57941	323	1183	30.46
	-5	67616	289	1594	36.70
	-6	86933	256	2315	48.53
SARK	-4	5806	33	698	4.97
	-5	7043	0	1074	6.34
	-6	10442	0	1680	9.52
	-7	16563	0	2692	15.26
	-8	25760	0	4243	23.77
GEAR	-1	3134	279	1470	15.57
	-2	3646	273	2013	17.18
	-3	3671	275	2615	20.26
	-4	4905	328	3572	27.95
	-5	6409	379	4900	37.33
ADAMS	-2	4101	-	2854	20.59
	-3	4094	-	2986	21.63
	-4	4948	-	4128	30.73
	-5	6348	-	5427	42.30

Table 6.16 : Problem p6.2, λ=5, a=100 (Figure 6.15)

Method & Order	Log_{10} Accuracy	FE	JE	Steps	CPU Time
ERK 3	-4	7224	-	2213	6.40
	-5	13803	-	4428	12.43
	-6	31321	-	10380	28.82
	-7	74315	-	24759	68.30
ERK 5	-4	5172	-	761	3.29
	-5	6925	-	1029	4.92
	-6	9855	-	1581	6.95
	-7	15143	-	2497	10.21
	-8	23984	-	3969	16.23
BRK 3	-2	15781	429	429	8.90
	-3	20836	371	671	12.11
	-4	29870	375	1252	20.47
	-5	54959	291	2674	37.14
BRK 5	-3	55767	569	989	27.14
	-4	57670	488	1026	30.14
	-5	64917	409	1239	31.67
	-6	88803	303	1970	45.09
SARK	-1	13835	427	461	10.13
	-2	16692	361	623	12.04
	-3	21066	375	1136	17.50
	-4	25604	318	1729	22.88
	-5	23776	180	2233	22.07
	-6	16360	0	2701	15.74
	-7	22952	0	3797	21.96
GEAR	-1	2219	232	988	9.54
	-2	2680	220	1461	13.06
	-3	2959	213	1924	15.97
	-4	3878	271	2781	22.37
	-5	4955	296	3806	30.01
ADAMS	-2	5468	-	3682	25.64
	-3	5944	-	4266	31.84
	-4	6700	-	5470	38.92
	-5	7479	-	6086	45.38

Table 6.17 : Problem p6.2, λ=10, a=100 (Figure 6.16)

Method & Order	Log_{10} Accuracy	FE	JE	Steps	CPU Time	OVHD
ERK 3	-5	29348	-	7616	28.17	
	-6	30920	-	8144	29.98	
	-7	37242	-	10275	36.45	
	-8	55983	-	16551	56.36	
ERK 5	-5	32254	-	5362	24.40	
	-6	32488	-	5401	24.37	
	-7	32960	-	5483	24.76	
	-8	34245	-	5695	25.84	
BRK 3	-2	4807	184	94	2.81	
	-3	5244	155	131	3.34	
	-4	6504	130	224	4.40	
	-5	8789	84	454	6.58	
	-6	13479	84	837	10.69	
BRK 5	-3	57255	511	1179	26.91	
	-4	47709	304	955	21.92	
SARK	-2	3714	125	177	2.79	1.61
	-3	5228	130	286	3.93	2.28
	-4	6775	139	317	4.82	2.68
	-5	10206	79	968	10.32	7.10
	-6	12898	91	790	10.75	6.68
GEAR	-3	470	45	224	2.02	1.87
	-4	628	48	351	2.92	2.72
	-5	863	59	531	4.28	4.00
	-6	1153	68	763	5.94	5.57
ADAMS	-4	45772	-	32220	203.34	
	-5	43673	-	33582	210.06	

Table 6.18 : Problem p6.3 λ=100, a=100 (Figure 6.17)

Method & Order	Log_{10} Accuracy	FE	JE	Steps	CPU Time
ERK 3	-2	18657	-	5876	18.95
	-3	51360	-	16711	53.03
	-4	133644	-	44140	138.50
	-5	338377	-	112463	352.22
ERK 5	-3	13843	-	2060	9.85
	-4	23010	-	3559	16.84
	-5	38787	-	6160	28.56
	-6	65495	-	10603	48.60
BRK 3	0	14937	18	1628	15.13
	-1	22373	9	2468	22.96
	-2	38548	8	4268	39.29
	-3	67154	7	7445	68.26
	-4	116785	10	12956	118.97
	-5	199011	11	22060	201.76
BRK 5	-2	10319	12	555	7.39
	-3	28089	11	1528	20.47
	-4	56543	8	3124	41.63
SARK	-3	13845	0	2060	12.44
	-4	23013	0	3560	20.70
	-5	38788	0	6160	35.43
	-6	65490	0	10602	59.75
GEAR	-2	3903	217	3158	23.51
	-3	5633	299	4726	34.07
	-4	8637	452	7408	52.96
	-5	13427	700	11641	82.60
ADAMS	-2	5859	-	3361	25.81
	-3	7517	-	4788	38.82
	-4	10717	-	7872	64.52
	-5	14222	-	11698	96.97

Table 6.19 : Problem p6.3 (Figure 6.18)

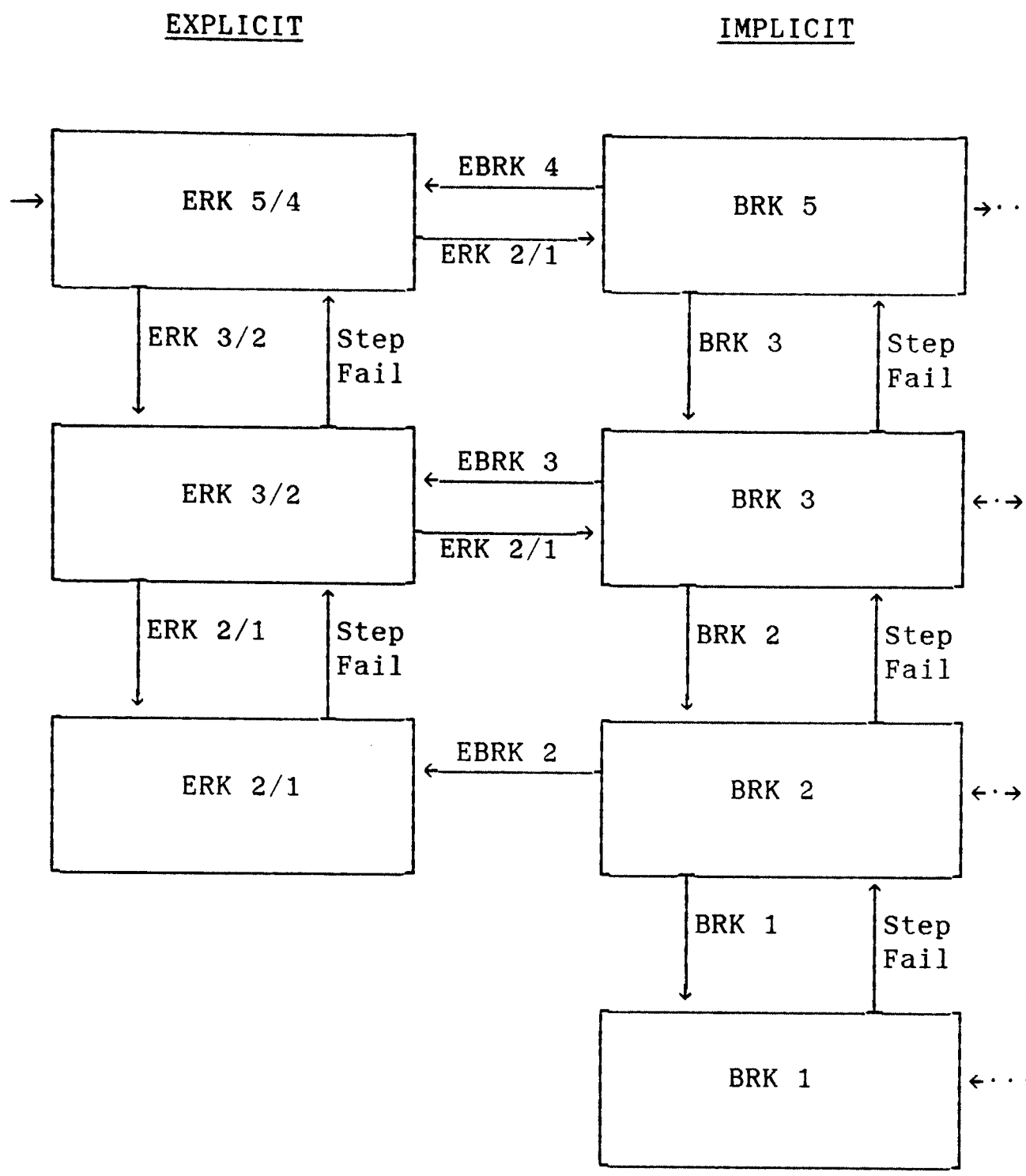


Figure 6.1: Schema of the complete algorithm

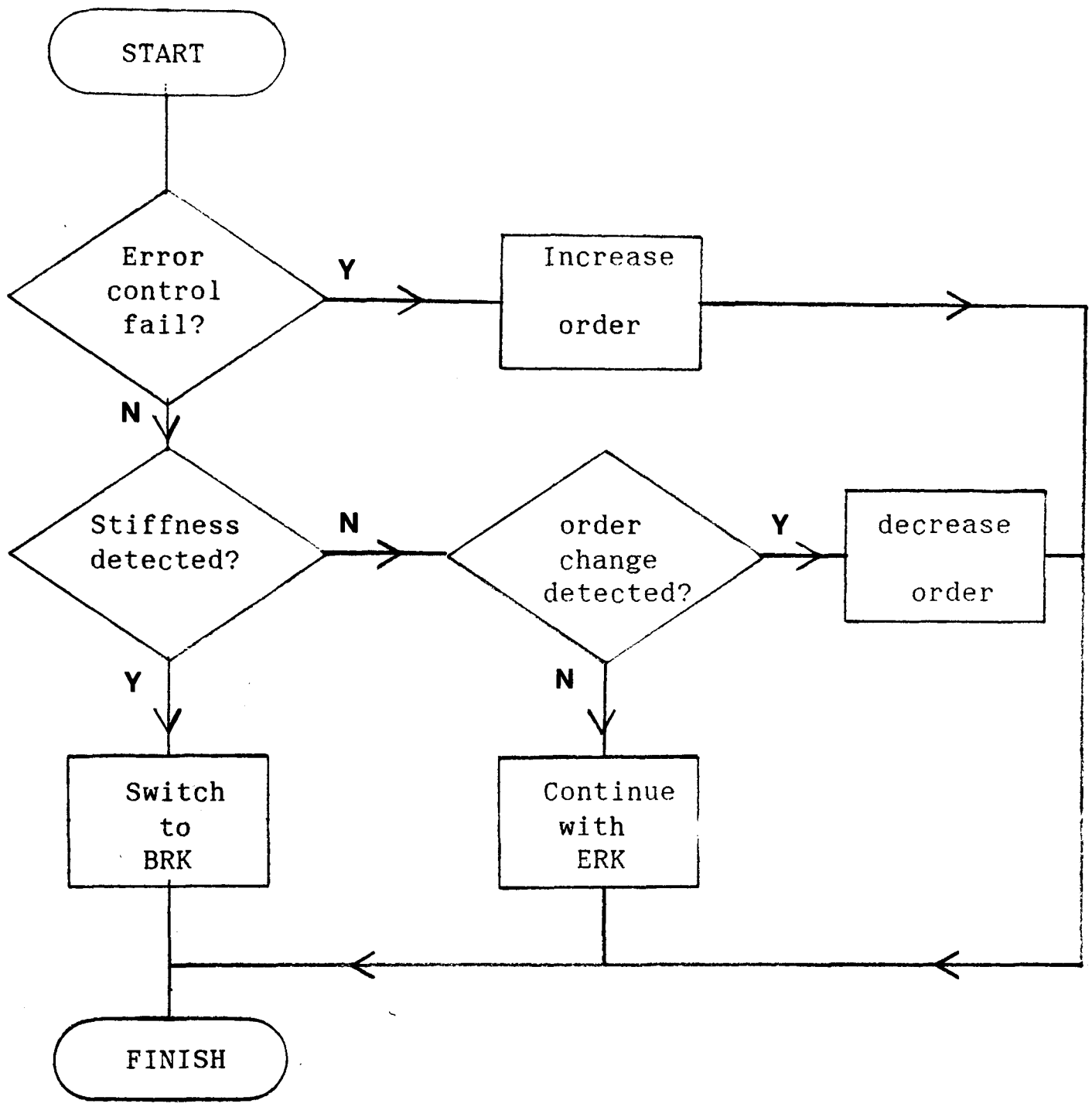


Figure 6.2 : Switching strategies for ERK3

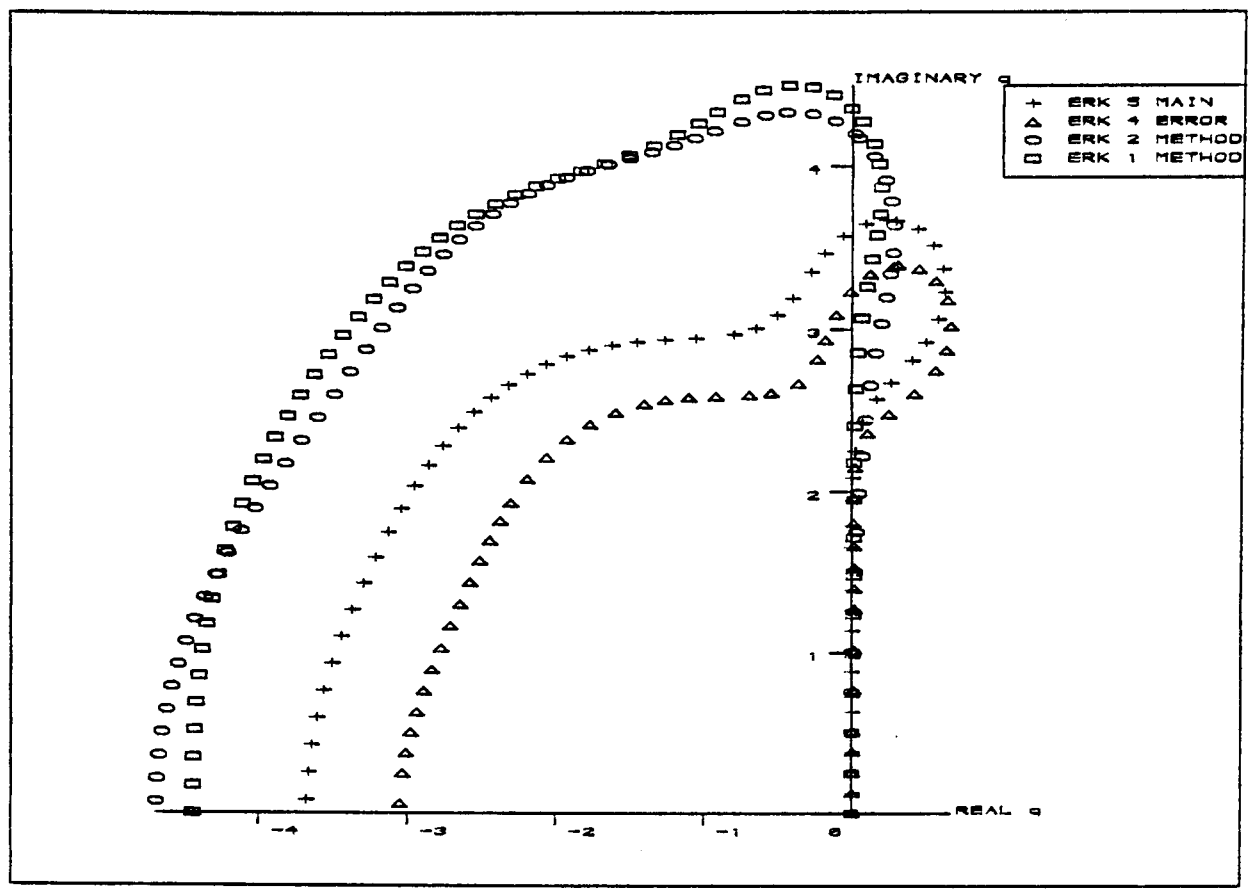


Figure 6.3 : 5th order Main ERK and embedded (2,1) pair.

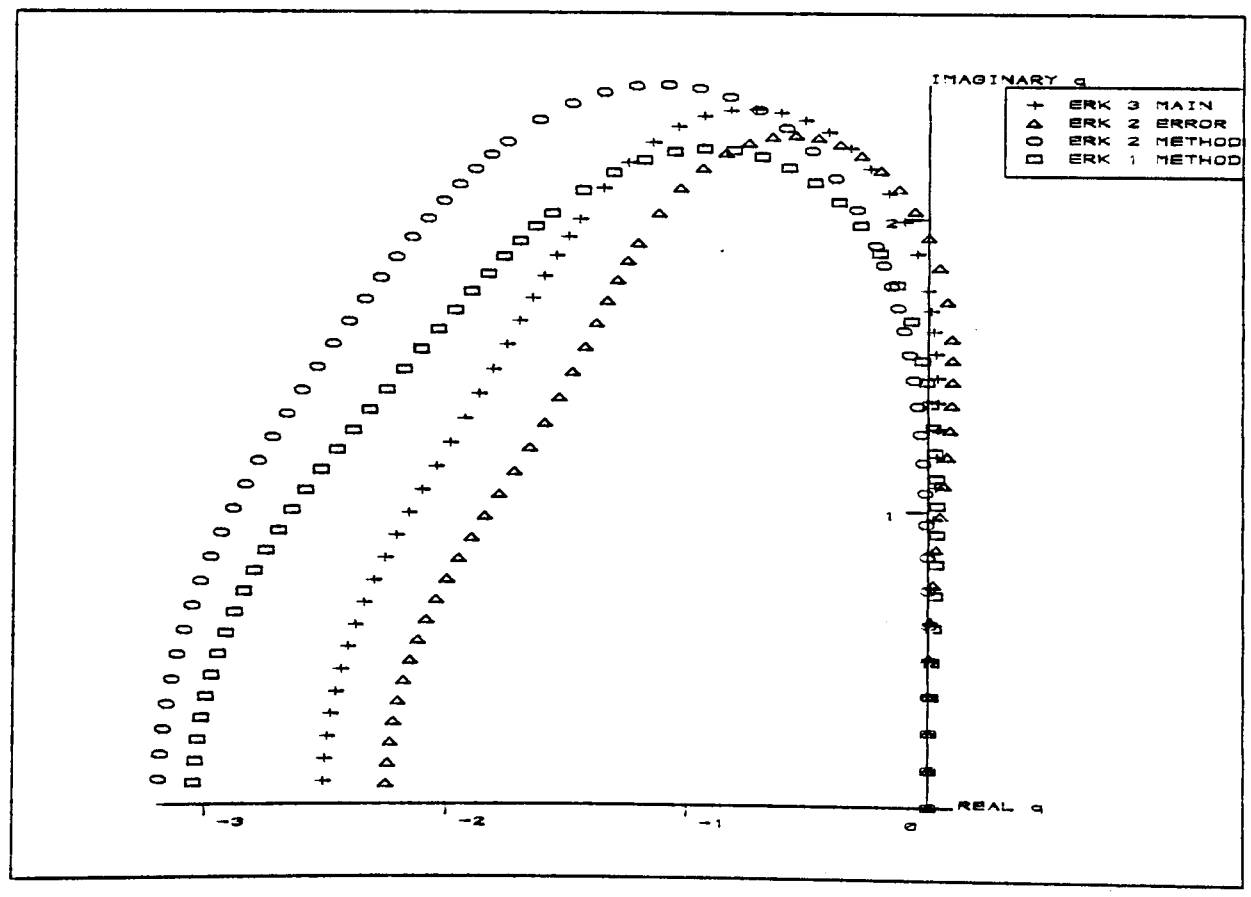


Figure 6.4 : 3rd order main ERK and embedded (2,1) pair.

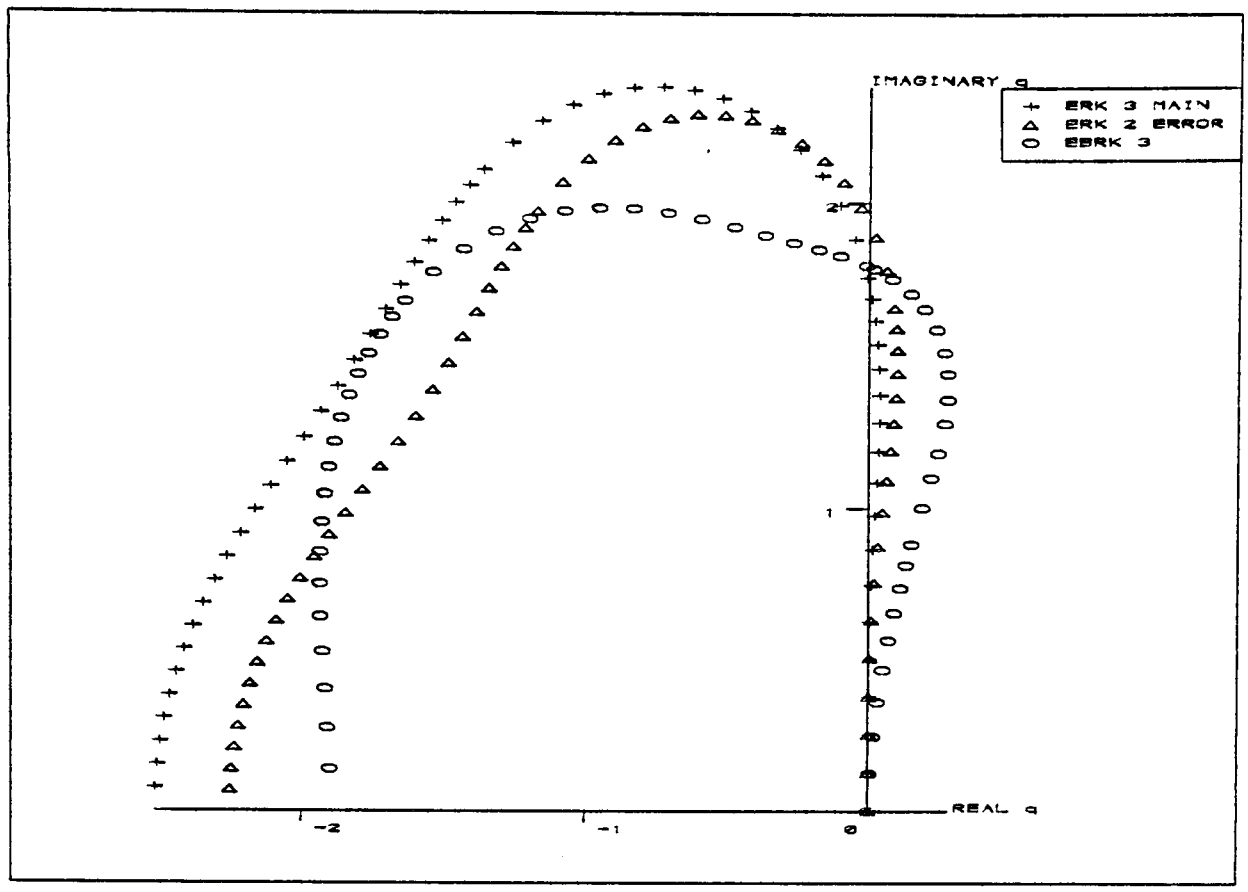


Figure 6.5 : Main ERK 3 and EBRK 3.

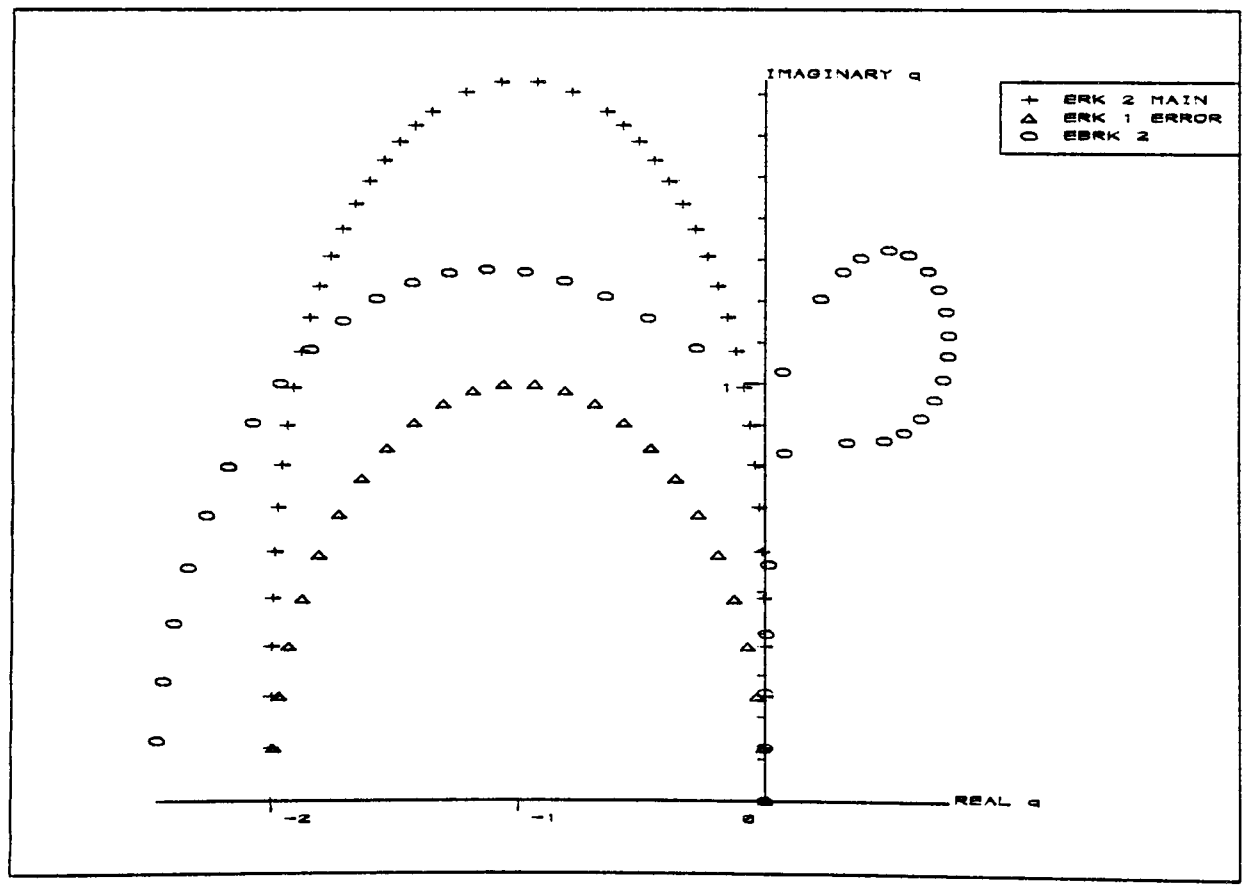


Figure 6.6 : Main ERK 2 and EBRK 2.

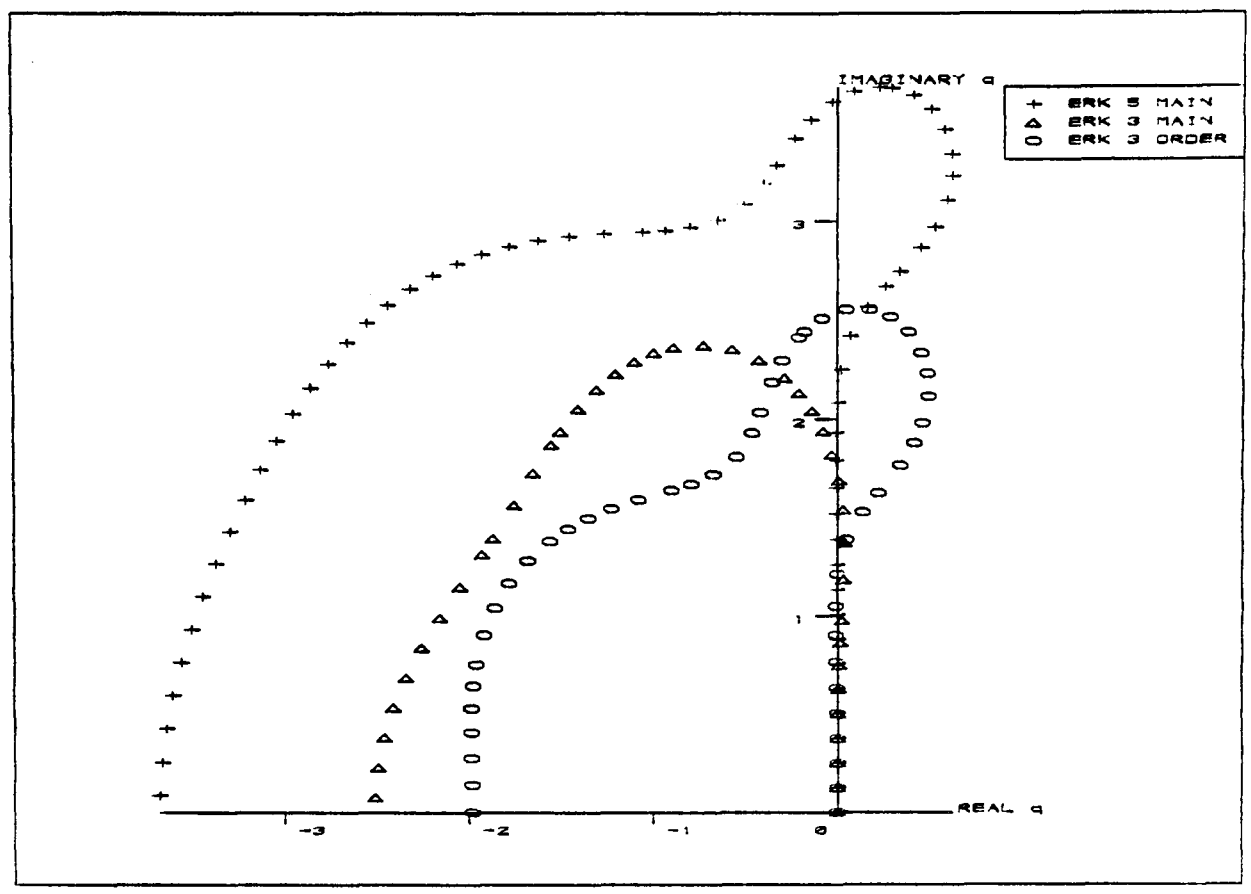


Figure 6.7 : ERK 5, ERK 3 and 3rd order embedded ERK.

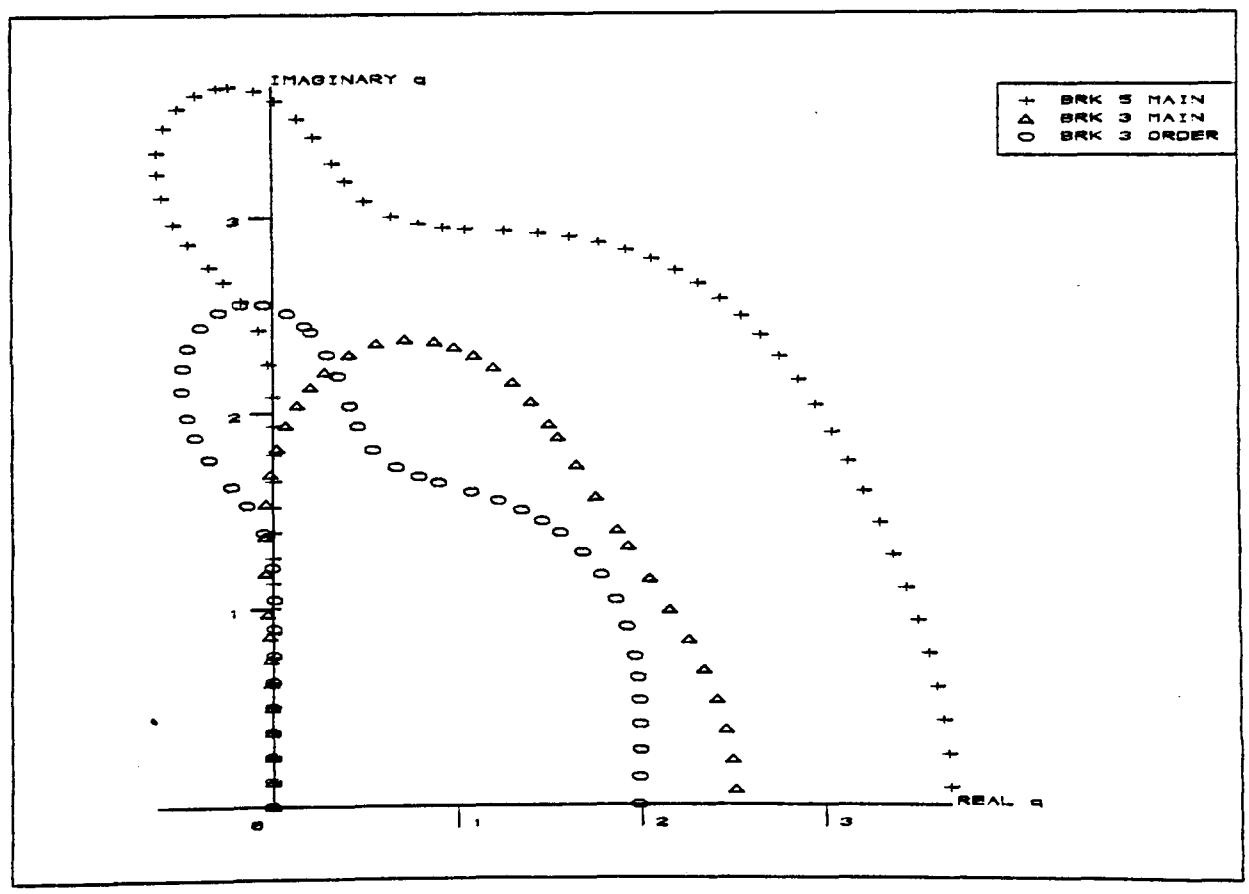


Figure 6.8 : BRK 5, BRK 3 and 3rd order embedded BRK.

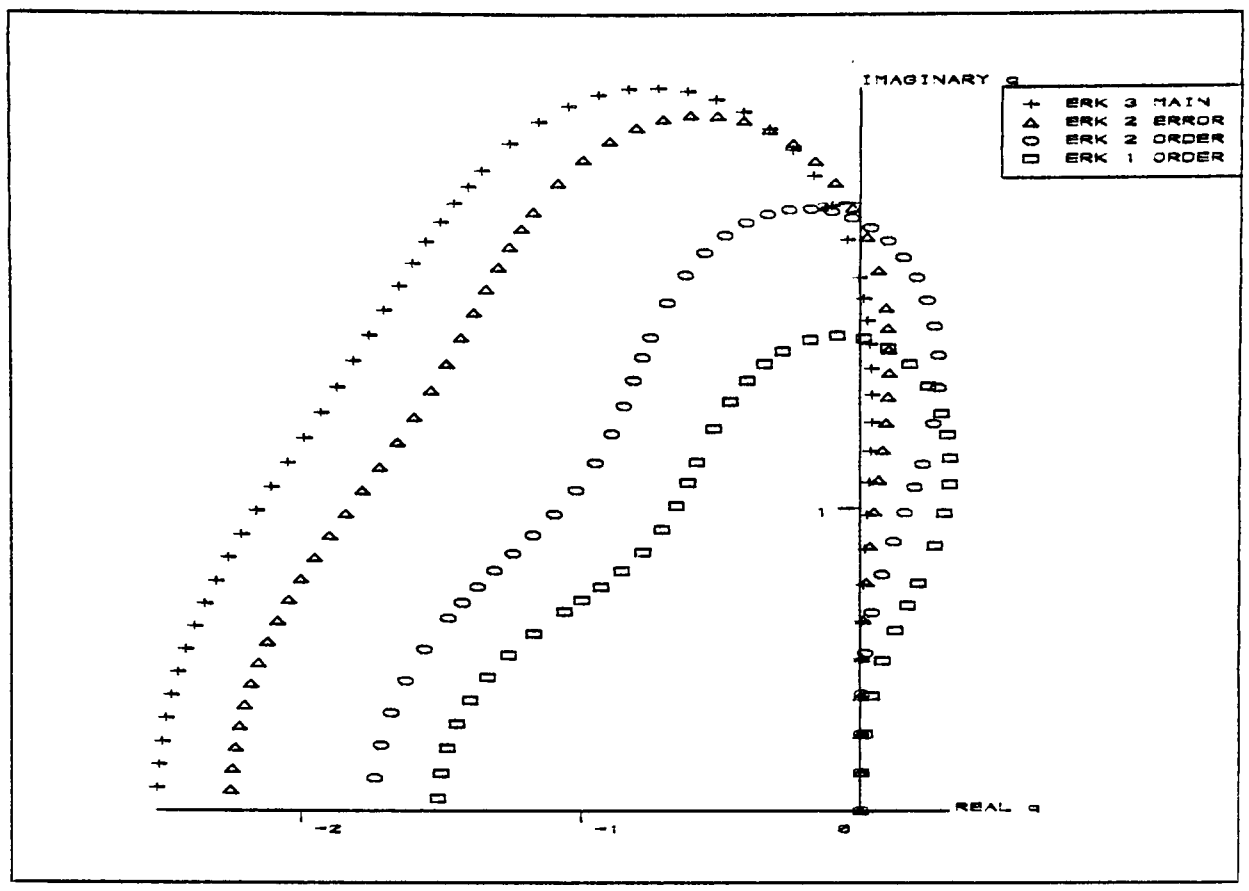


Figure 6.9 : ERK 3 and 2nd order embedded ERK.

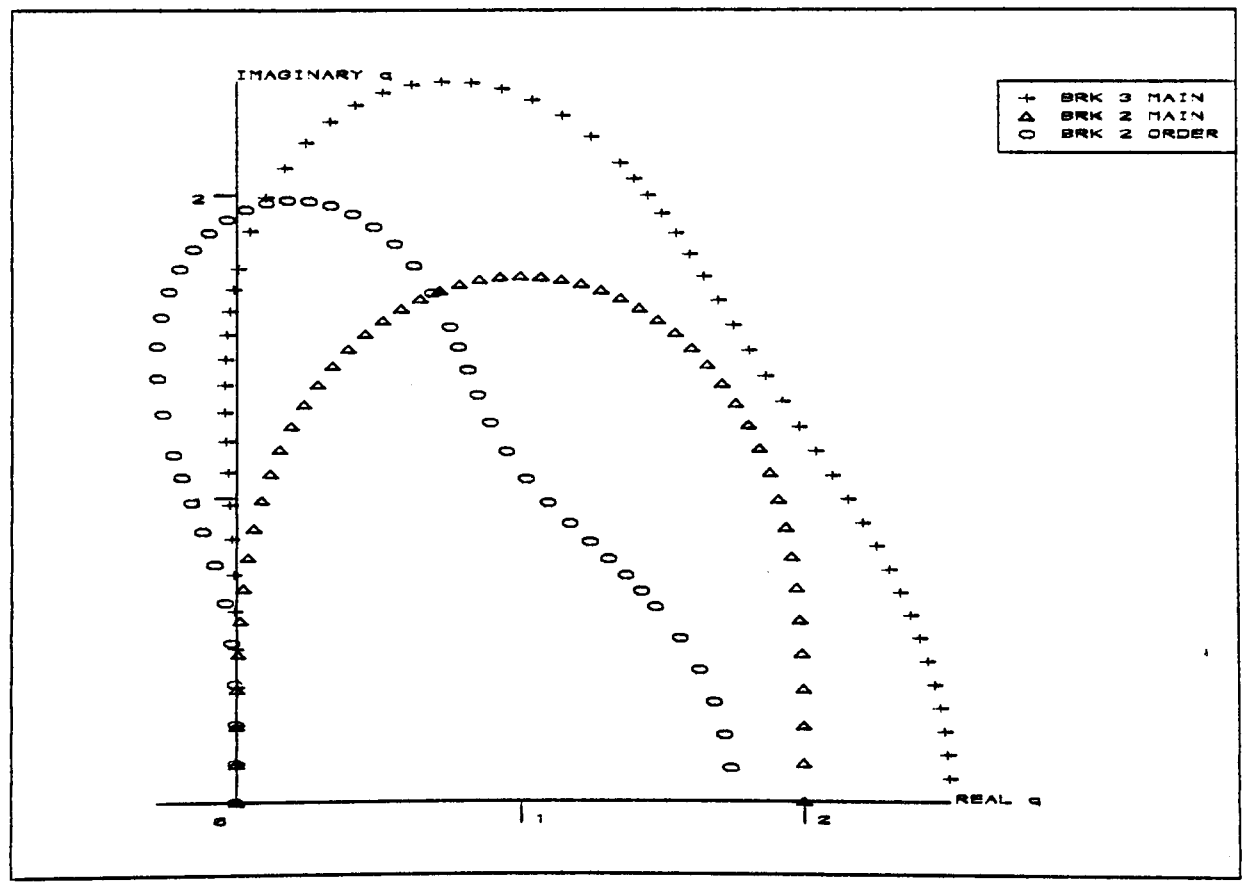


Figure 6.10 : BRK 3, BRK 2 and 2nd order embedded BRK.

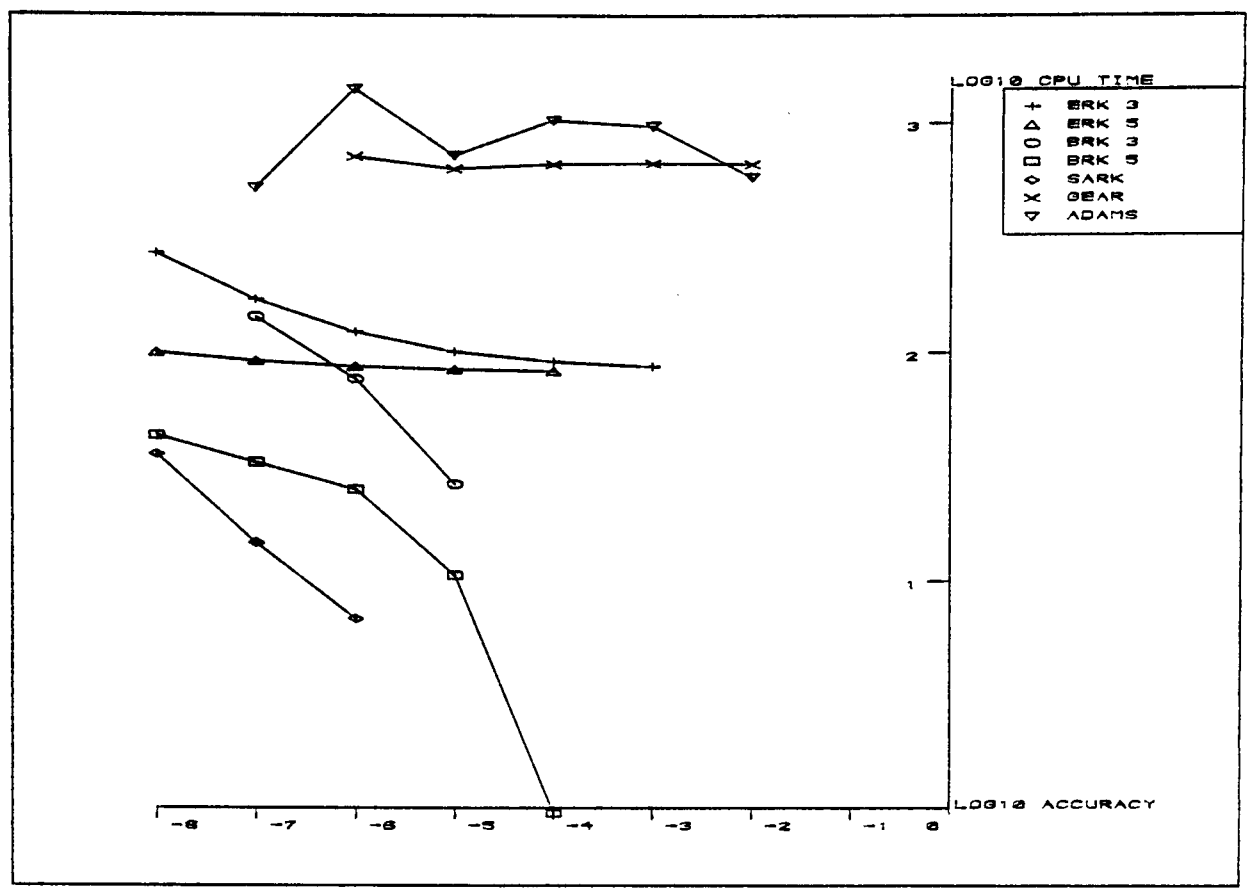


Figure 6.11 : Problem p6.1.

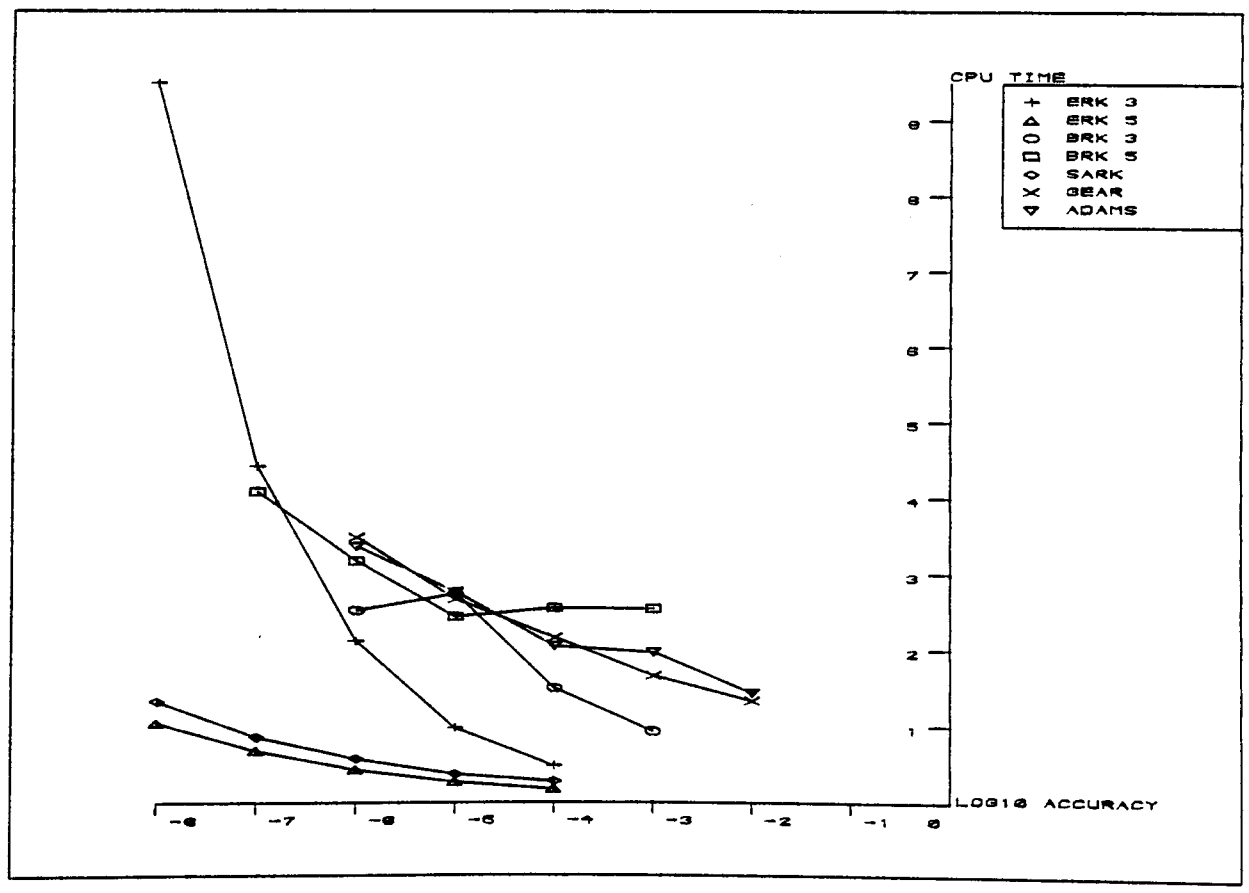


Figure 6.12 : Problem p6.2 $\lambda = 5$, $a = 10$.

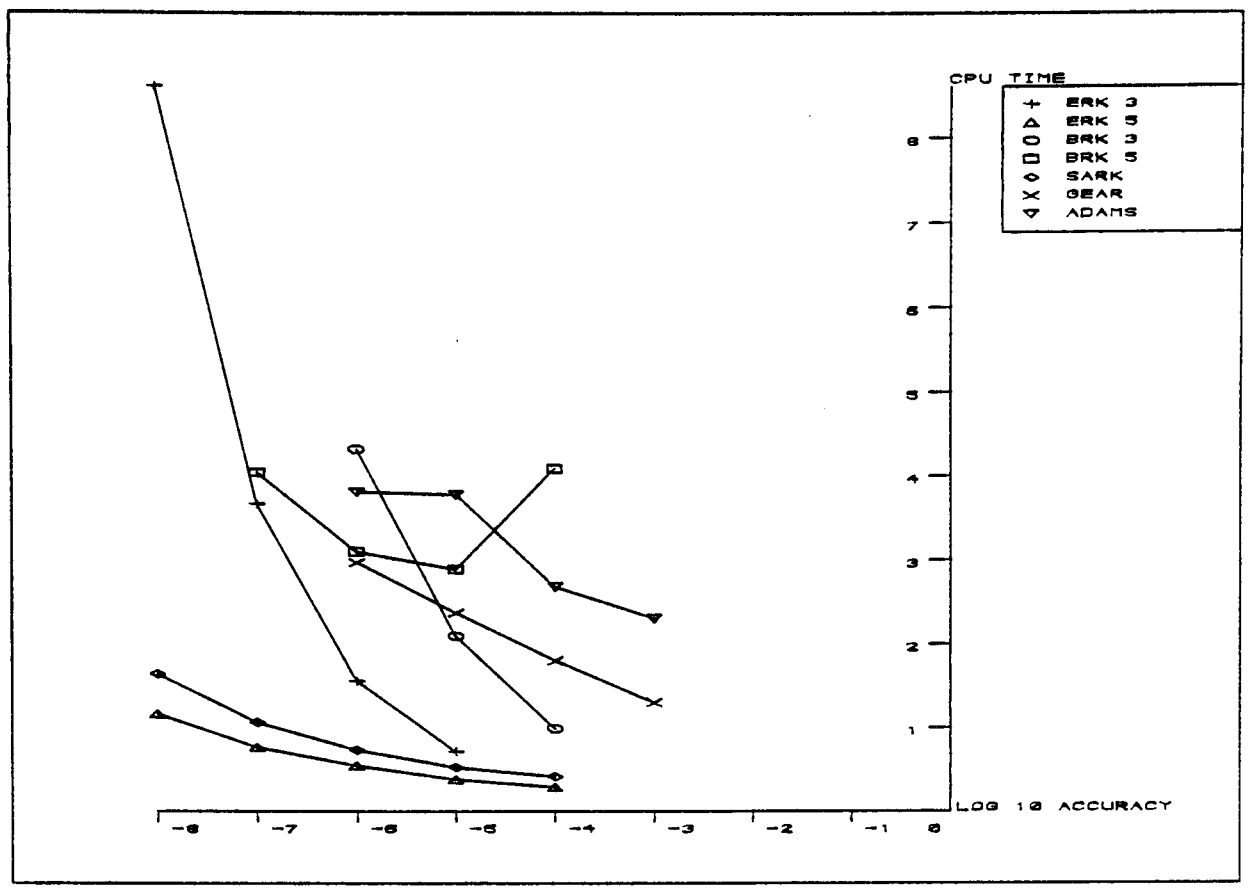


Figure 6.13 : Problem 6.2 λ = 10, a = 10.

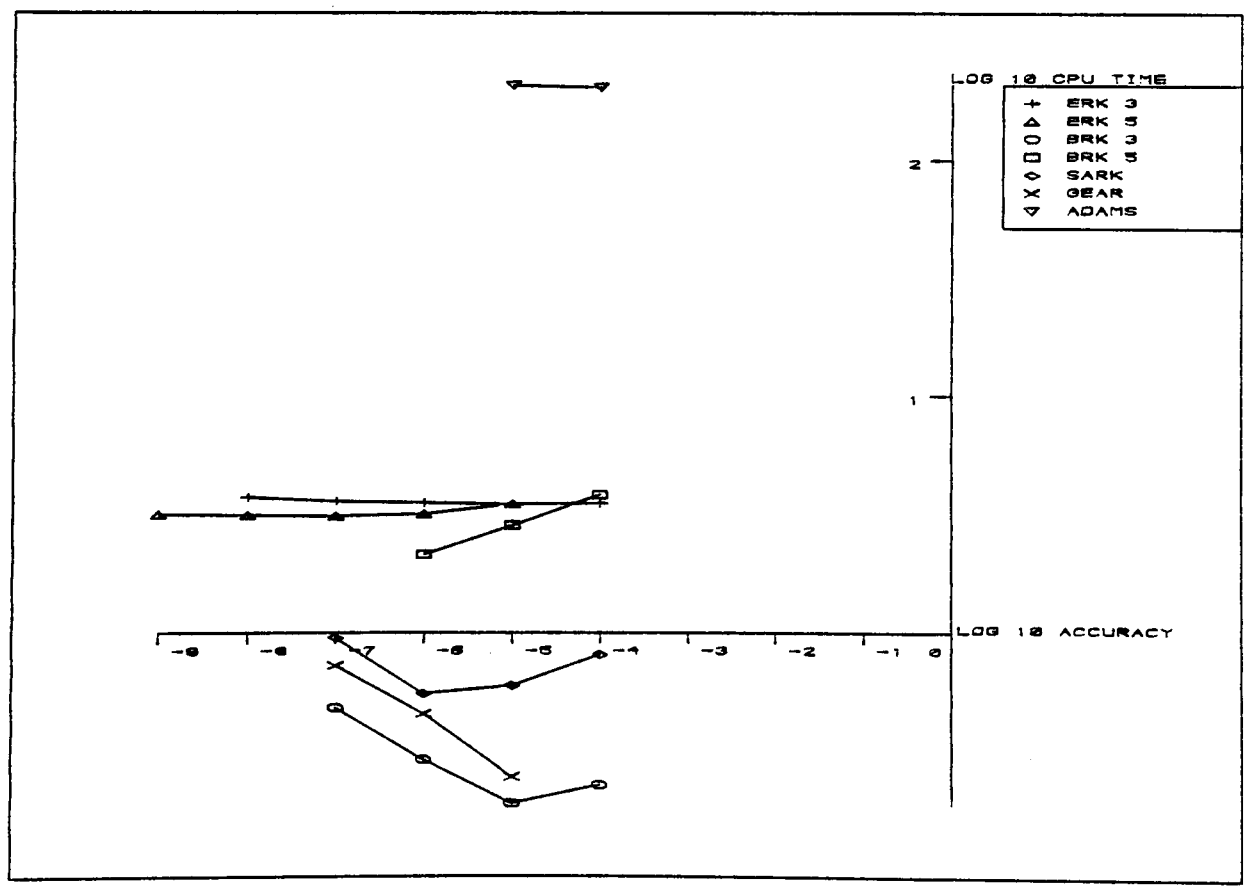


Figure 6.14 : Problem p6.2 λ = 100, a = 10.

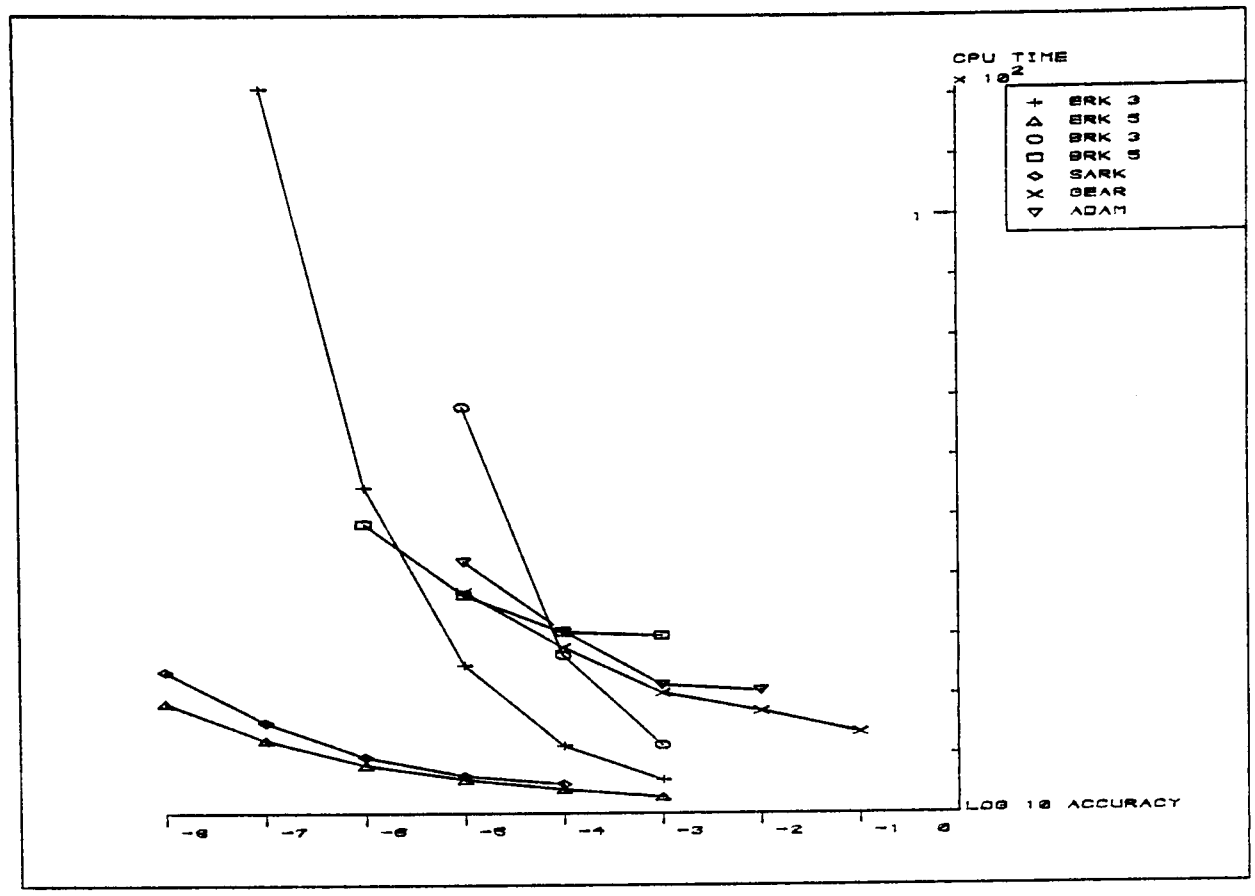


Figure 6.15 : Problem p6.2 $\lambda = 5, a = 100$.

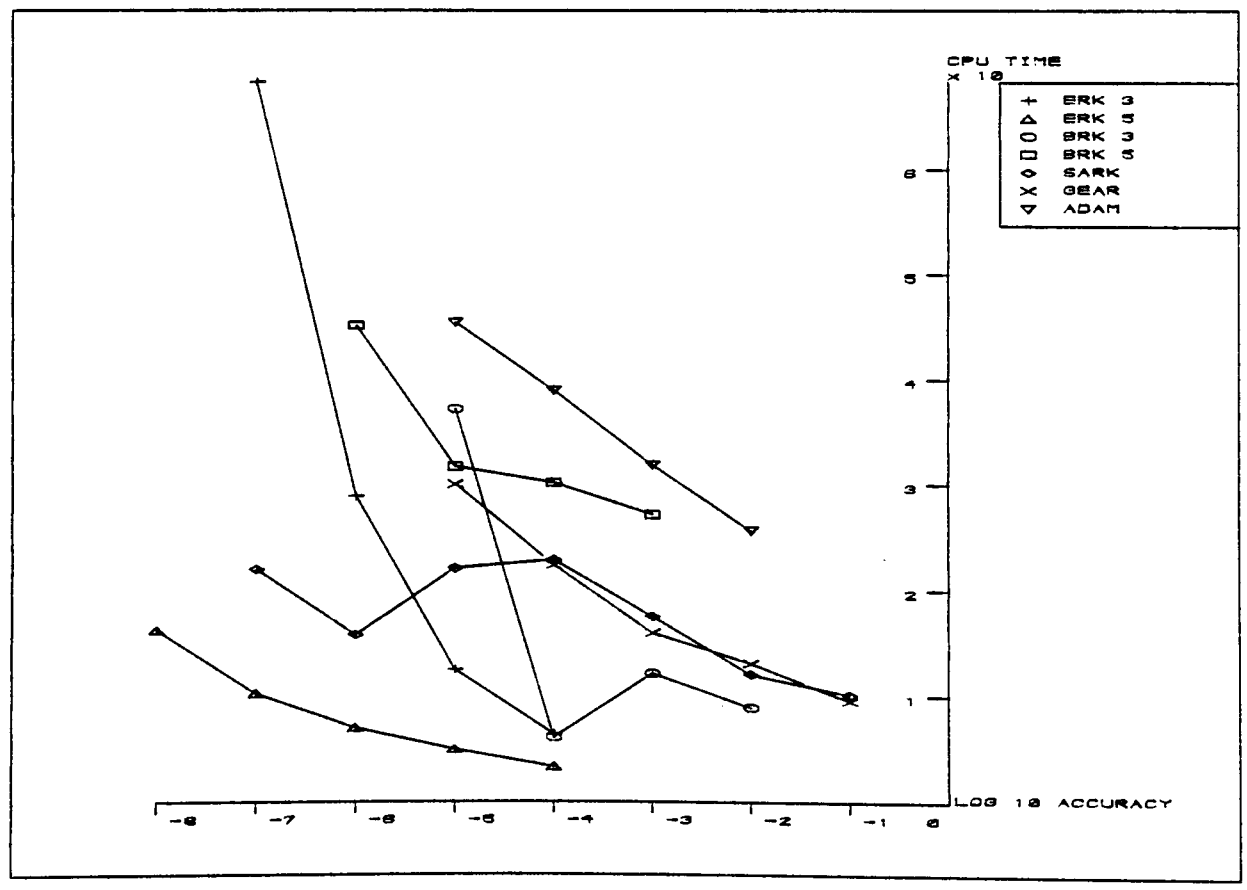


Figure 6.16 : Problem p6.2 $\lambda = 10$, $a = 100$.

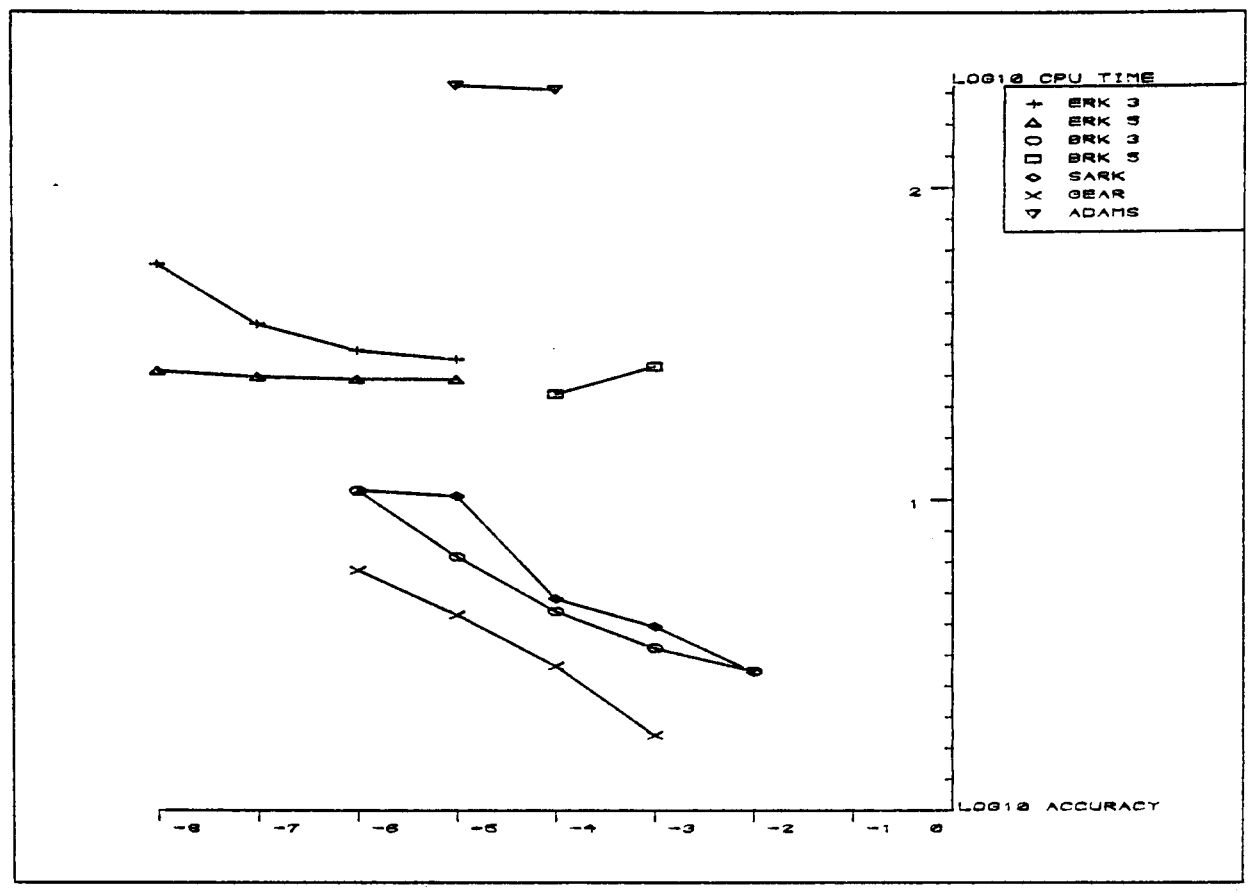


Figure 6.17 : Problem p6.2 $\lambda = 100, a = 100$.

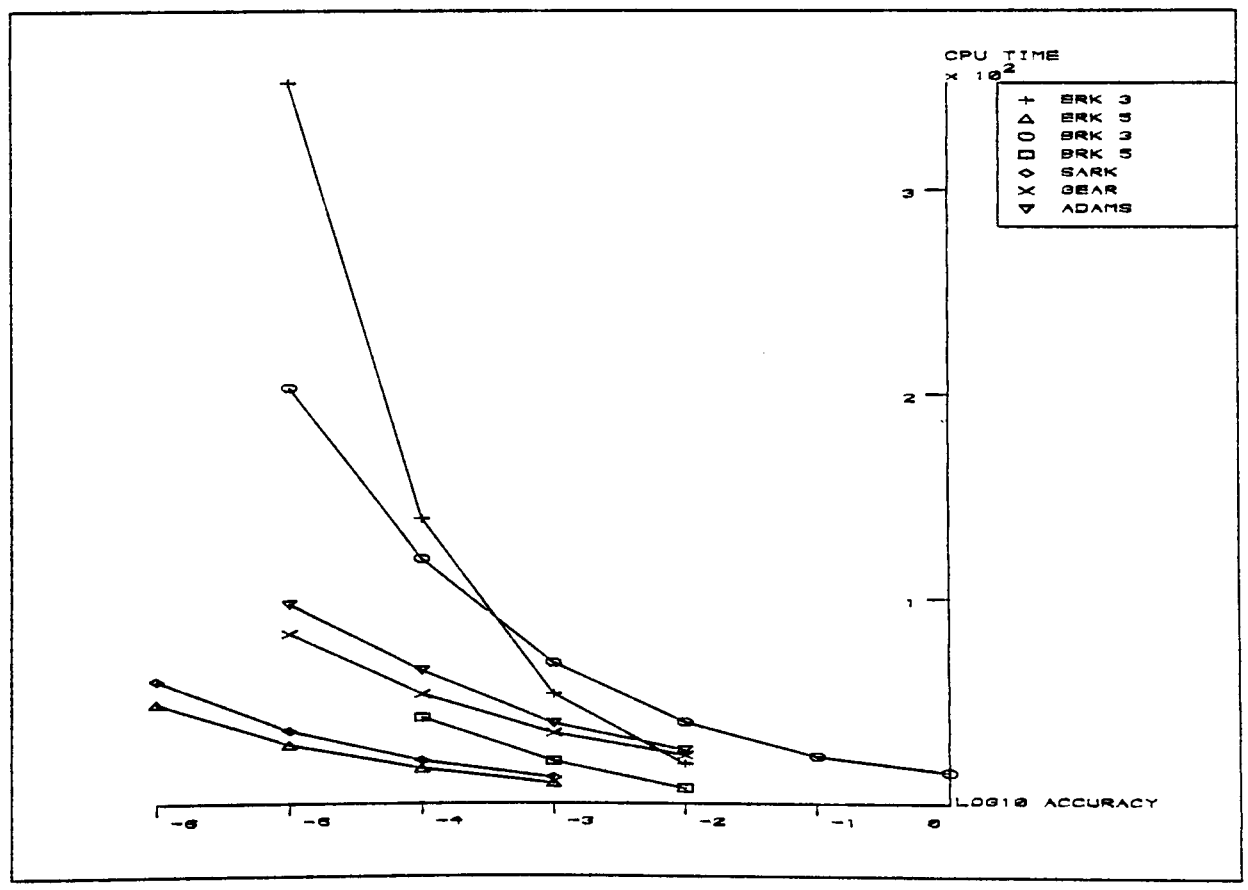


Figure 6.18 : Problem p6.3

Chapter 7 : NUMERICAL COMPARISIONS

When a new numerical method is developed it must be rigorously tested on a wide variety of problems to ensure that its performance is satisfactory. By "satisfactory", we do not necessarily mean that it should be compared directly against some other code to ensure that it can always perform better. Often the new code will tackle an area not covered extensively by any existing code. Alternatively the new code may be inferior to the old one for a particular class of problems for which the old one is specifically designed. However, it is better over a broader spectrum of problems. SARK falls into this category. We expect it to perform less well than a specialist non-stiff solver on non-stiff problems, and less well than a stiff solver on stiff problems. SARK should, however, perform better than these specialist codes when tested over a mixed collection of problems. The codes selected were the NAG implementation of the ADAMS code, D02QAF, and the GEAR code, D02QBF.

The aim of this chapter is to compare SARK with two existing highly developed extensively used codes over an unbiased test set. The standard test package available for testing ODE solvers is that of Enright and Pryce[1983], DETEST. This package comprises two problem sets, one stiff and the other non-stiff. SARK was expected to perform reasonably well on both sets of problems, but to be beaten on the non-stiff problems by the ADAMS code and be beaten on the stiff problems by the GEAR code. However, SARK performed rather better than this.

We can attempt to rank these codes using a number of criteria eg.

i) storage requirements,

ii) CPU time or

iii) compactness of code.

In terms of iii) SARK is some 1000 lines of FORTRAN code whereas GEAR and ADAMS are typically twice this amount. When considering array storage, however, SARK will be worse. GEAR requires one vector of size N, one array of N^2 and one array of size 22×N. SARK on the otherhand requires 18 vectors of size N, two arrays of N^2 and various smaller vectors of dimension up to 10. The remainder of this chapter tackles the ranking according to CPU time.

7.1 The problems considered

The problems considered in this chapter are all taken from the DETEST package. This is a package that is designed to assess the performance of a numerical method over a selected set of problems. It has been widely used for this purpose, eg. Petzold[1983], Norsett and Thomsen[1986] and many more.

The package is split into two sections, one containing stiff problems and the other non-stiff. Each section is split into a number of groups, with each group having a common theme. For the non-stiff section the groups considered are

A : Single equations

B : Small systems (2-3 state variables)

C : Moderate systems (10-51 state variables)

D : Orbit equations

E : Higher order equations

and for the stiff section the groups are

A : Linear with real eigenvalues

B : Linear with non-real eigenvalues

C : Non-linear coupling

D : Non-linear with real eigenvalues

E : Non-linear with non-linear eigenvalues

Each group contains up to 6 separate problems. A complete list of all the problems can be found in Enright and Pryce[1983].

The method being assessed is timed over selected problems for a variety of specified tolerance values. For each problem chosen a series of statistics are produced that depends upon the choice of two parameters; one controls how detailed the results are and the other controls the calculation and tabulation of normalized efficiency statistics.

The first option can produce either global error results at the endpoint of the integration or the maximum observed global error throughout the integration range. The maximum global error is assessed by using an internal integrator, but using a tighter tolerance than requested for the timed method to compute the "true" solution. For the non-stiff section this internal routine is the DVERK code of Hull et al.[1977] and for the stiff section the SECDER code of Addison[1980]. The maximum global error is then calculated as

$$\max_{x\in[0,x_{end}]} \max_{i=1,..N} \|y_i - y_i(x)\| \qquad (7.1)$$

where the vector norm is the maximum norm.

As stiff problems usually pass through two phases, ie. an initial transient phase and a steady state phase, it appears natural to monitor the maximum global error rather than just the endpoint global error, thus the maximum global error is used throughout this chapter. Unfortunately the internal integrator for the stiff section is occasionally unable to complete enough successful integrations for

global errors to be assessed adequately. When this happens no normalized statistics can be produced for that particular problem.

The methods being tested were executed on each problem considered at tolerances of 1.e-3, 1.e-4, 1.e-5, 1.e-6, 1.e-7 and 1.e-8 and tabulated results produced. As each method is not directly controlling the same quantity, when a tolerance level is selected, a direct comparison of these results is profitless.

We assume that the numerical method is attempting to keep

$$\text{global error} \approx C \times TOL^{E} \qquad (7.2)$$

where the exponent, E, and the constant of proportionality, C, depend upon the method and the problem. After determing the value of the global error for each prescribed tolerance the value of C and E are determined. These can then be used to define the expected accuracy as a function of the tolerance, TOL. This is the accuracy a user can expect when a problem is solved with a specified tolerance of TOL. It is then possible to tabulate cost against expected accuracy. This is performed in the package by using the normalized option. Thus different methods can be compared in an unbiased manner. These tables are reproduced in Appendix A, for the non-stiff cases and Appendix B, for the stiff cases. Thus in Table B.1, for stiff problem A1, to achieve a maximum global error of 1.e-4, SARK requires 0.57 seconds and GEAR 1.66 seconds.

This chapter is not intended to solely compare GEAR, ADAMS and SARK directly as they all clearly tackle different problem areas. It is rather intended to determine which method should be used for a particular category of problem. It is also demonstrated that for most

problems a type-insensitive code is a valid alternative to a specialized code.

7.2 Non-stiff problems

The tabulated results of Appendix A are shown graphically in Figures 7.1 - 7.25. The headings used for each column are the same as those used in the previous chapter with $\log_{10}$ accuracy being the expected accuracy as stated above. Each figure shows the CPU time against this expected accuracy for each method used to solve the non-stiff problems, viz. SARK, GEAR and ADAMS.

7.2.1 Group A : Figures 7.1 - 7.5

The problems in this group have functions which are relatively inexpensive to evaluate and hence the Runge-Kutta based code is very efficient. Clearly SARK integrates with the explicit method throughout, indicated by no Jacobian evaluations. On the whole SARK is approximately 2.5 times faster than ADAMS.

7.2.2 Group B : Figures 7.6 - 7.10

Here function evaluations are more expensive yet SARK is able to perform much better than either GEAR or ADAMS.

7.2.3 Group C : Figures 7.11 - 7.15

Problems C4 and C5 highlight the effect of selecting the incorrect integrator for a non-stiff system. GEAR is particularly inefficient on these two problems which have 51 and 30 state variables respectively. Even though Runge-Kutta based codes are known to perform inefficiently when function evaluations are computationally expensive, SARK is able to perform better than ADAMS on these problems.

7.2.4 Group D : Figures 7.16 - 7.20

This group contains one basic non-linear system with a free parameter that is adjusted to form each problem. As the parameter is increased the problems become more demanding. SARK clearly performs better than ADAMS and GEAR on every setting of this parameter, with problem D5 being particularly demanding for the multistep methods.

7.2.5 Group E : Figures 7.21 - 7.25

This group consists of high order equations reduced to a system of first order equations. Problem E2 is van der Pol's equation with λ = 1. Not surprisingly SARK performs well on this problem.

7.2.6 Summary of non-stiff results

Clearly comparing numerical methods is a very difficult process, especially if a large number of problems are involved, Table 7.1 summarises the results of Appendix A by summing up the CPU times over tolerances that are common to each method, eg. for problem A1 the total CPU time is accumulated over expected tolerances 1.e-4, 1.e-5, 1.e-6 and 1.e-7. The most efficient method being the one with the smallest total CPU time. Totals over each group are also shown. Taking the set as a whole SARK is approximately 2.5 times quicker than ADAMS and 6.5 times faster than GEAR. These results show that GEAR is particularly inefficient for non-stiff problems. In fact SARK out performs GEAR and ADAMS on every single problem and never switches over to the implicit method. It is likely, however, that the ADAMS code will be the most efficient for some problems were the function evaluations are very expensive.

The type-insensitive codes of Petzold and Norsett and Thomsen both use the non-stiff set of problems to demonstrate their codes. Both incorrectly diagnose some of the non-stiff problems as stiff. Petzold[1983] claims, "Very few of the problems of non-stiff DETEST were diagnosed as stiff", and Norsett and Thomsen[1986] produce results for problems A4, A5, B5, D1, D2 and D5 that require Jacobian evaluations. This indicates that the stiff mode of solution was employed for part of the integration range.

7.3 Stiff problems

When a problem is encountered which is known to be stiff a non-stiff integrator would not be employed. Hence there is little point in evaluating the performance of ADAMS over this section of DETEST. The tabulated results of Appendix B are shown graphically in Figures 7.26 - 7.47.

7.3.1 Group A : Figures 7.26 - 7.29

Generally SARK is able to integrate this group of problems faster than GEAR, except problem A2. This is a problem which has nine state variables and function evaluations are expensive to evalute.

7.3.2 Group B : Figures 7.30 - 7.34

This group is classified by the problems having an oscillatory component. As shown in chapter 2, this type of problem is in fact incorrectly classified as stiff. The performance of SARK clearly verifies this, as it often performs the integration with the explicit method only.

The poor performance of GEAR on oscillatory problems has motivated many

modifications to the basic BDF methods, eg. Blended linear multistep methods (BLM) Skell and Kong[1977], Extended BDF methods (EBDF) Cash[1980] and Modified EBDF methods (MEBDF) Cash[1983]. All these methods have been compared with GEAR over a selection of problems including one oscillatory problem. The MEBDF methods are shown, Cash[1983], to perform marginally better than the BLM methods on problem B5 of DETEST. In the same paper MEBDF methods are shown to be better, by a factor of between 1 and 7, than GEAR on problem B5 and 1.1 times faster over the remainder of the test group. SARK on the otherhand is between 6 and 15 times faster than GEAR on problem B5 and over 3 times faster on the remaining problems of the group.

7.3.3 Group C : Figures 7.35 - 7.39

Clearly when using a type-insensitive code to solve a problem there is a breakeven point where both integrators will be equally efficient. When this point is encountered the algorithm may produce erratic results, this is seen on problem C1. SARK has difficulty in determining which integrator is more efficient, although whichever it uses it still performs better than GEAR.

7.3.4 Group D : Figures 7.40 - 7.43

On all the problems in this group GEAR is able to perform better than SARK. This is not due to deficiencies in the switching strategies but rather to the nature of the implicit methods used. This is illustrated by the high number of Jacobian evaluations required by SARK. Repeated problems with singular Jacobian forces SARK to use a low order implicit method whereas GEAR can use relatively high order methods.

7.3.5 Group E : Figures 7.44 - 7.47

This group is split fairly evenly between the two methods, SARK performs better on E1 and E2 and GEAR on E3 and E4. Neither method was accurate enough to produce sufficient statistics for a comparison of problem E5 to be made. Unfortunately the internal integrator, SECDER, was unable to integrate problem E6 successfully enough to produce a "true" solution.

7.3.6 Summary of stiff results

Again a summary table is produced, Table 7.2, similar to that produced for the non-stiff section. For certain problems GEAR is clearly superior. However, taking an overall view of the stiff section, SARK is approximately 1.5 times faster than GEAR, even though a specilized stiff solver would be expected to perform best.

Problem Identifer	No. of tolerances	Total CPU time SARK	GEAR	ADAMS
A1	4	0.43	1.94	1.36
A2	4	0.27	1.98	1.01
A3	4	1.57	5.77	2.83
A4	4	0.24	1.54	0.92
A5	3	0.16	1.26	0.58
Total		2.67	12.49	6.70
B1	2	1.68	9.46	3.69
B2	4	1.01	4.75	3.45
B3	4	0.81	4.22	2.19
B4	5	3.15	12.02	5.54
B5	4	2.71	11.40	5.19
Total		9.36	41.85	20.06
C1	4	2.14	13.98	6.53
C2	4	5.58	15.43	13.58
C3	4	2.09	13.25	8.13
C4	3	5.99	114.16	20.79
C5	2	1.59	25.93	3.35
Total		17.39	182.75	52.38
D1	4	2.74	10.89	5.50
D2	4	2.57	14.40	5.80
D3	4	3.94	23.54	10.89
D4	4	4.46	28.73	12.23
D5	2	2.94	21.45	8.17
Total		16.65	99.01	42.59
E1	4	2.24	6.07	2.55
E2	5	6.34	24.50	9.97
E3	5	4.34	17.84	6.35
E4	2	0.11	0.85	0.33
E5	3	0.16	2.18	0.79
Total		13.19	51.44	19.99
Overall Total		59.26	387.54	141.72

Table 7.1 : Summary of non-stiff results

Problem Identifer	No. of tolerances	Total CPU time SARK	GEAR
A1	3	2.63	6.79
A2	1	5.45	3.88
A3	2	1.98	4.56
A4	1	4.23	6.00
Total		14.29	21.23
B1	3	19.28	22.99
B2	4	2.96	14.12
B3	4	3.76	15.09
B4	4	5.51	17.58
B5	4	17.84	179.84
Total		49.35	249.62
C1	4	3.96	9.03
C2	3	3.22	7.59
C3	4	6.37	8.59
C4	2	3.29	5.19
Total		16.84	30.31
D1	2	33.87	1.47
D2	2	11.33	2.23
D3	2	46.14	3.53
D6	1	6.83	0.35
Total		98.17	7.68
E1	3	1.66	3.09
E2	2	0.77	4.82
E3	3	21.14	4.06
E4	2	11.87	9.54
Total		35.44	21.51
Overall Total		214.09	330.35

Table 7.2 : Summary of stiff results

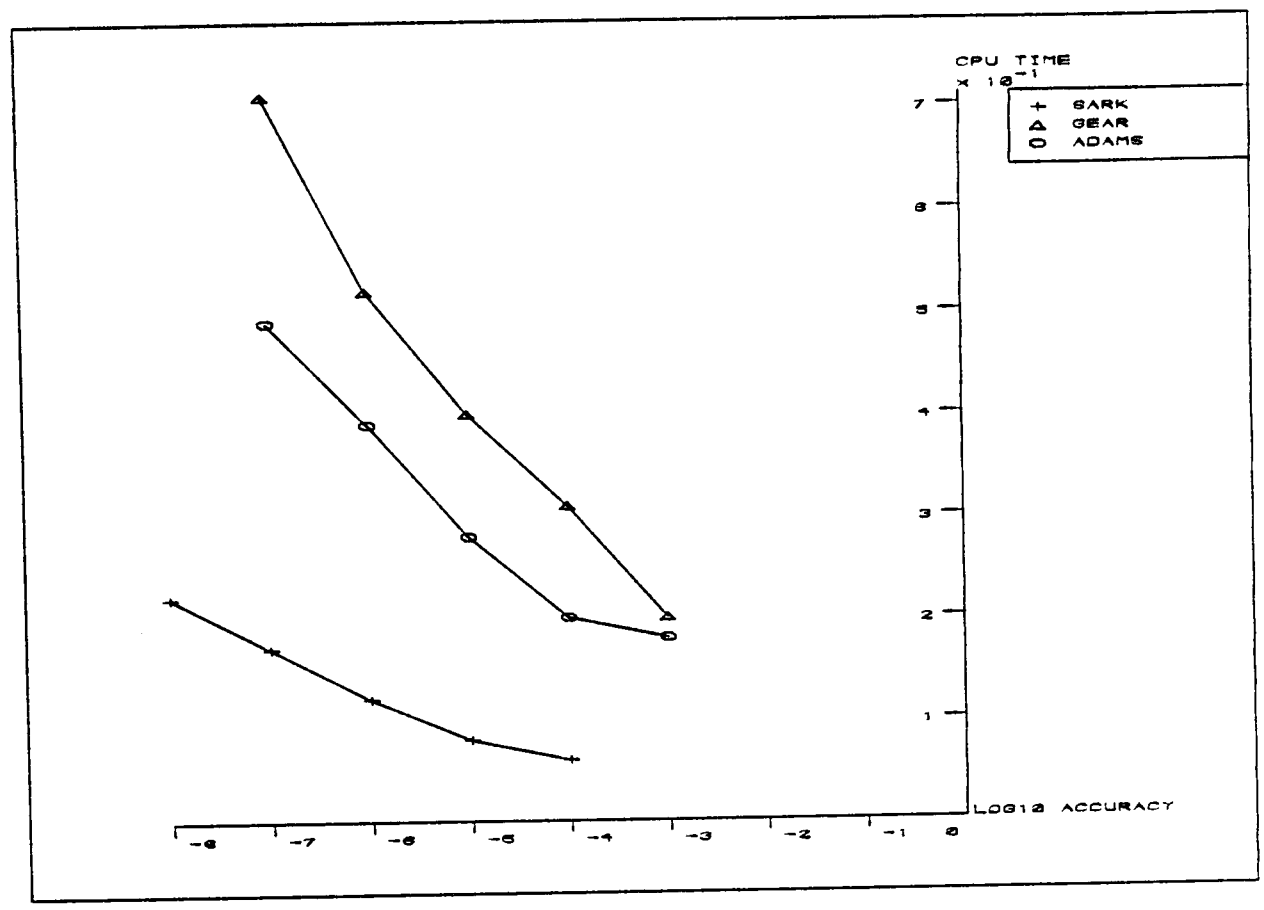


Figure 7.1 : Non-stiff problem A1

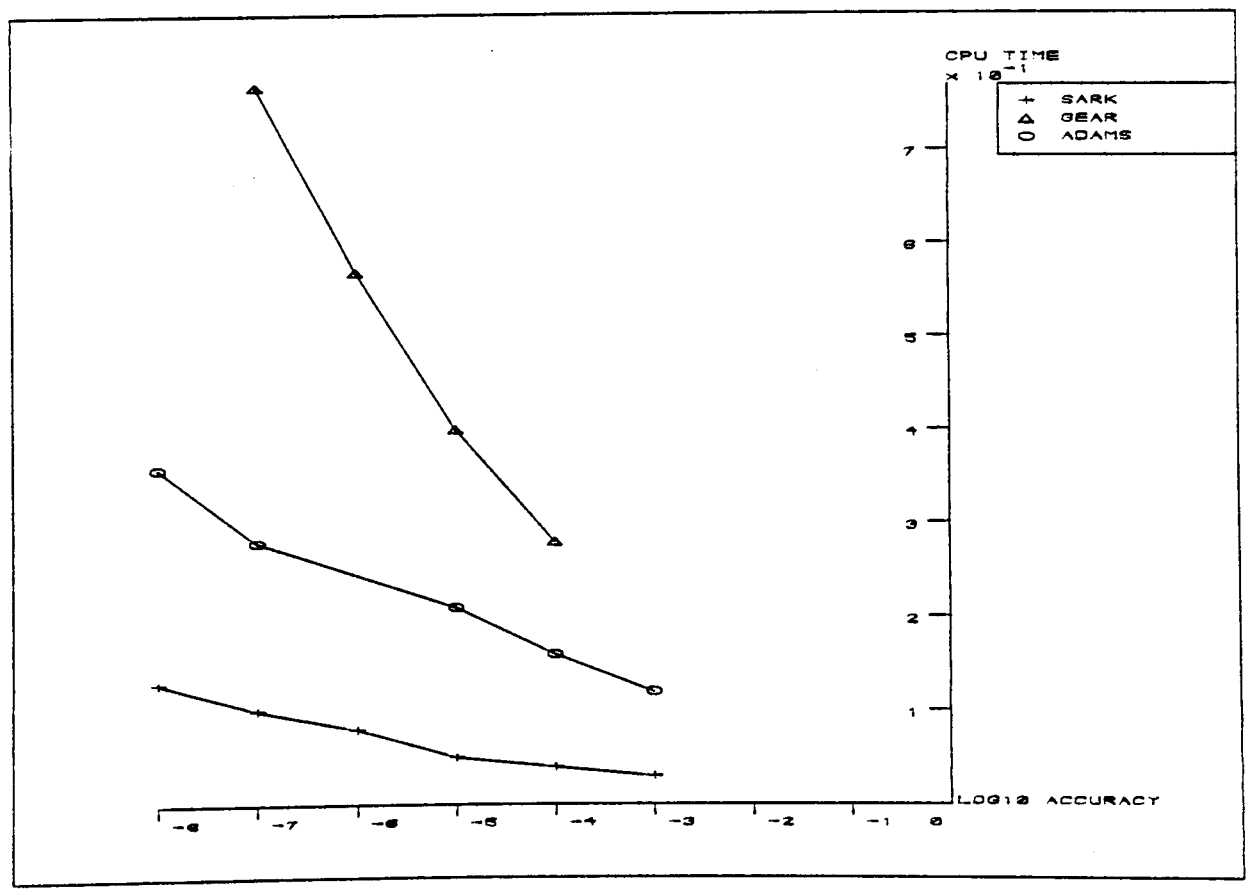


Figure 7.2 : Non-stiff problem A2

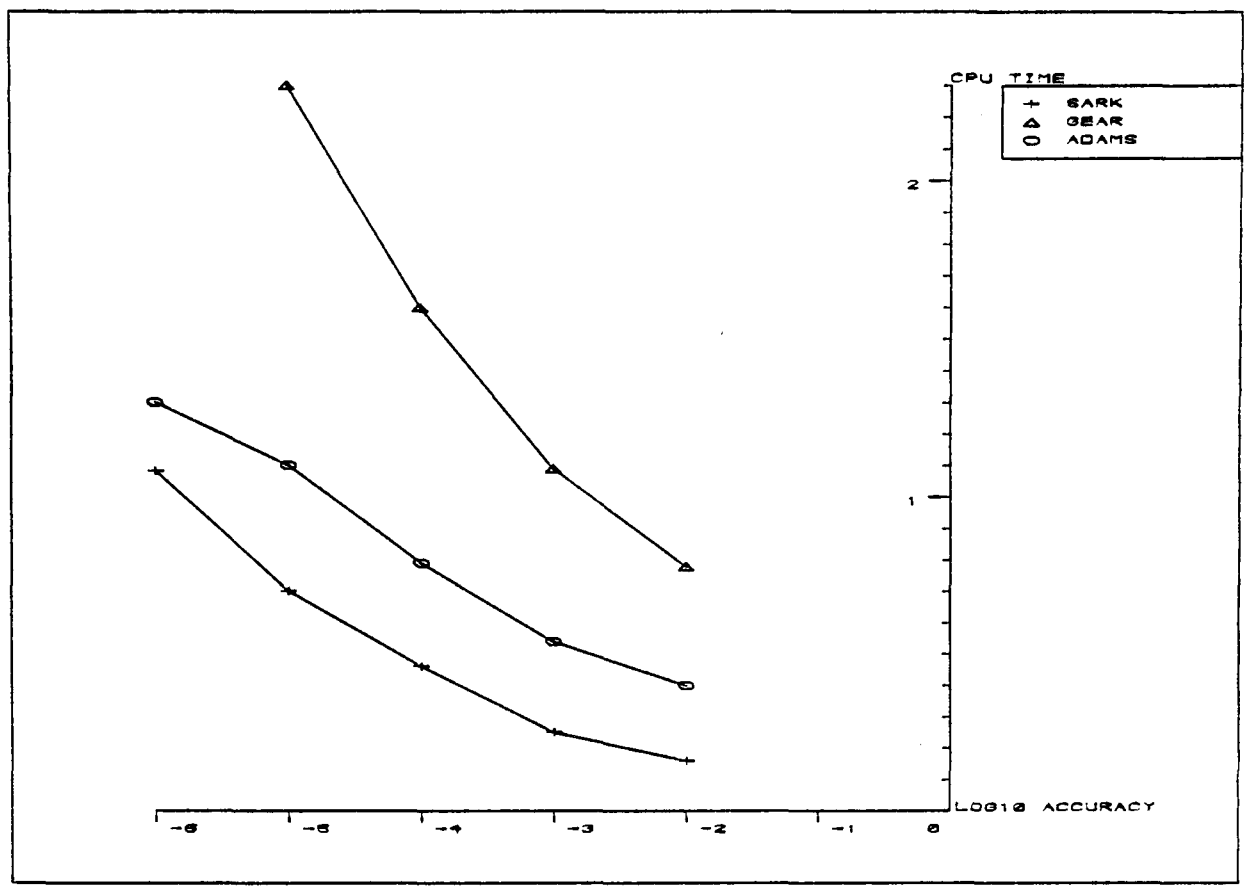


Figure 7.3 : Non-stiff problem A3

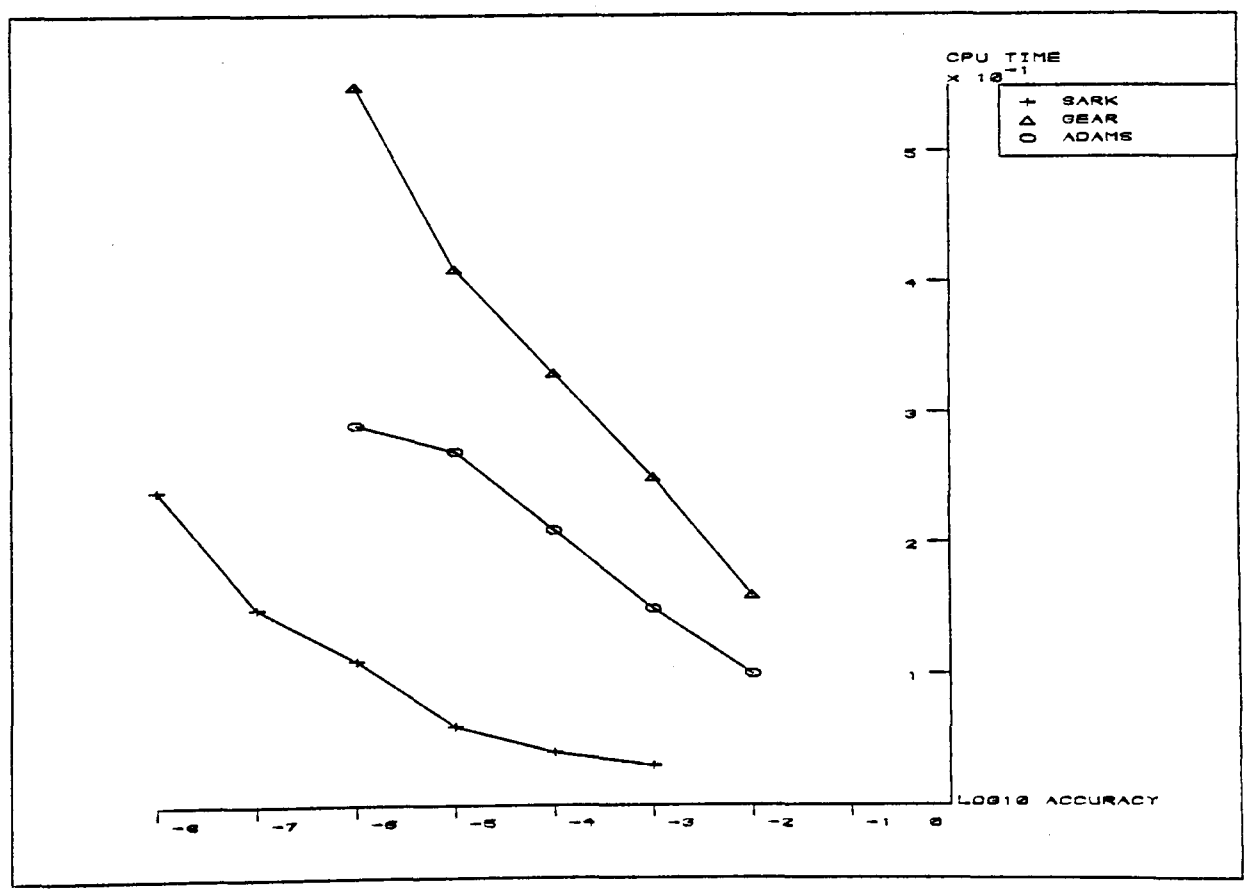


Figure 7.4 : Non-stiff problem A4

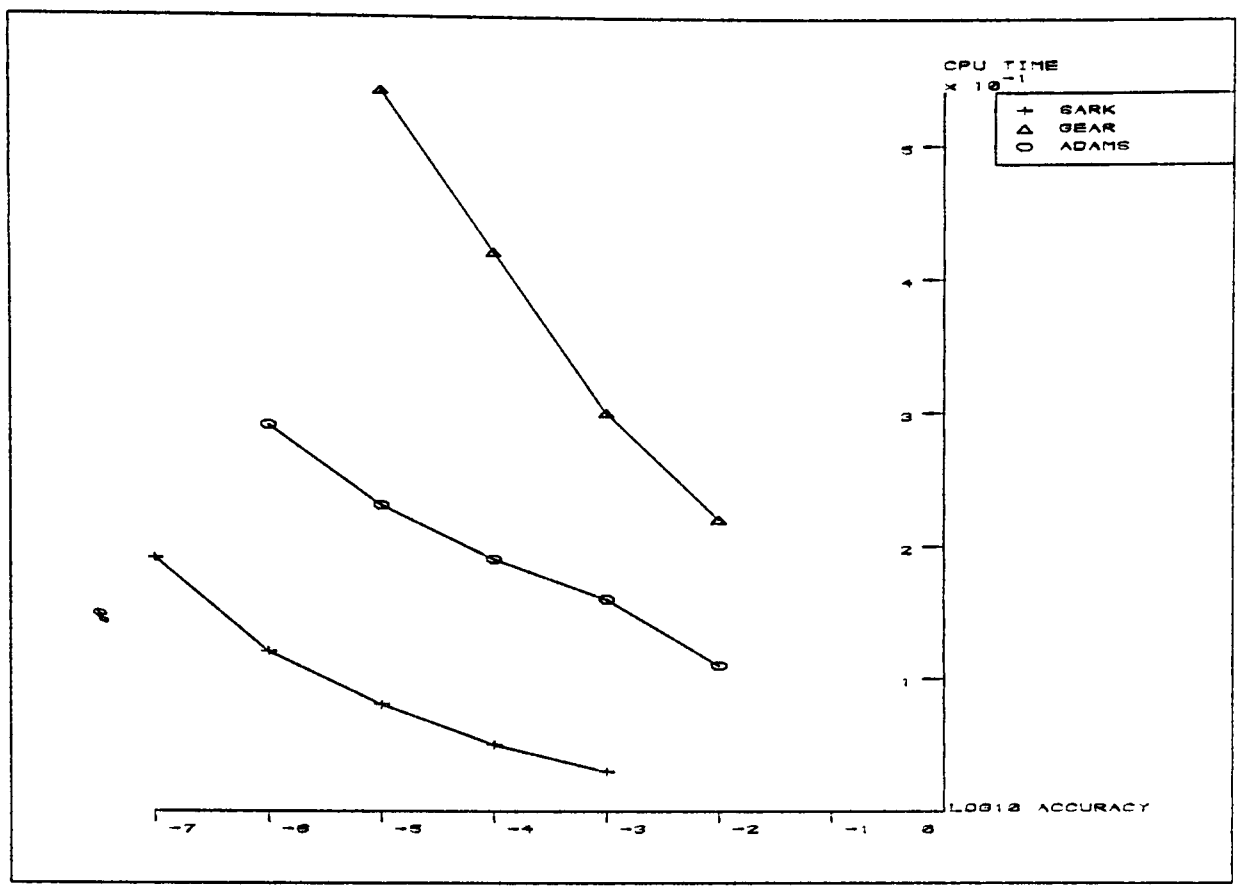


Figure 7.5 : Non-stiff problem A5

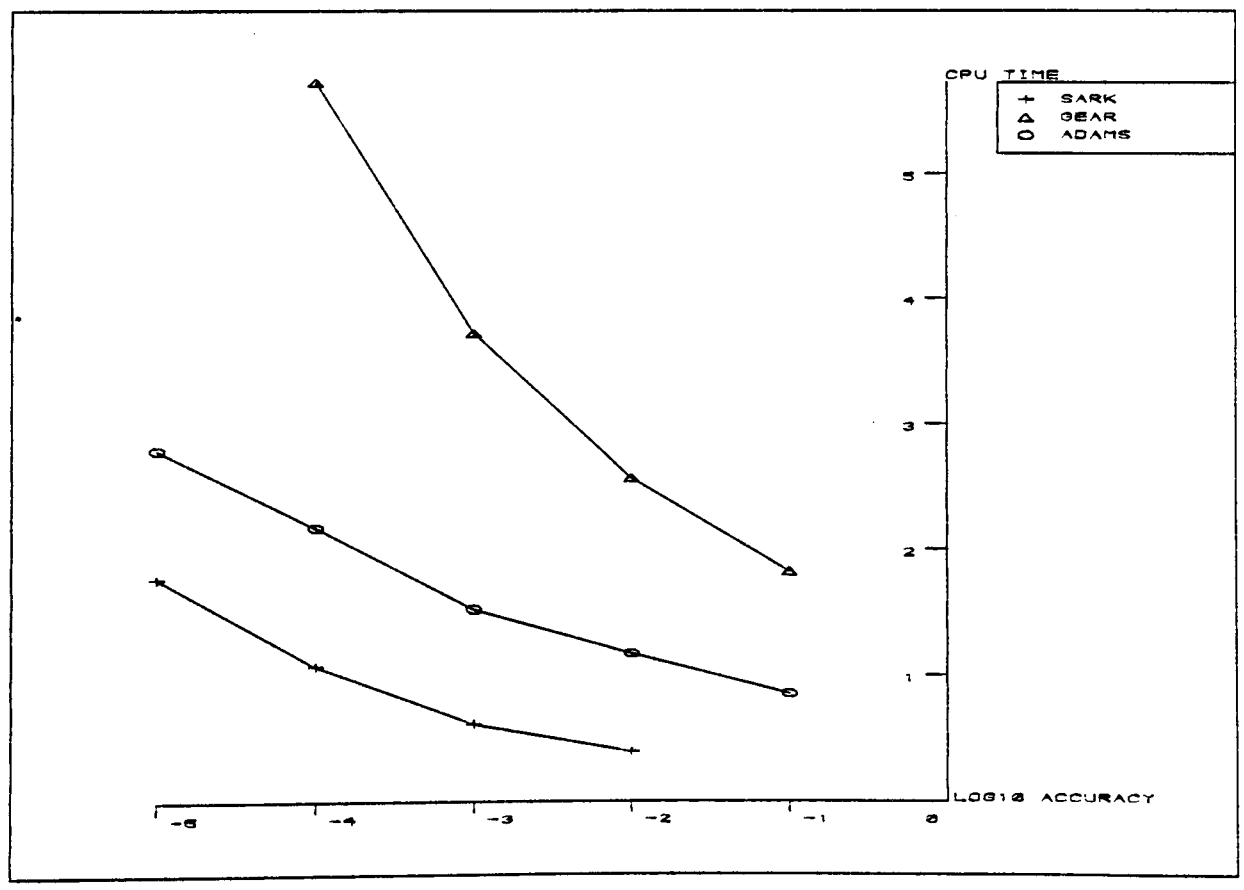


Figure 7.6 : Non-stiff problem B1

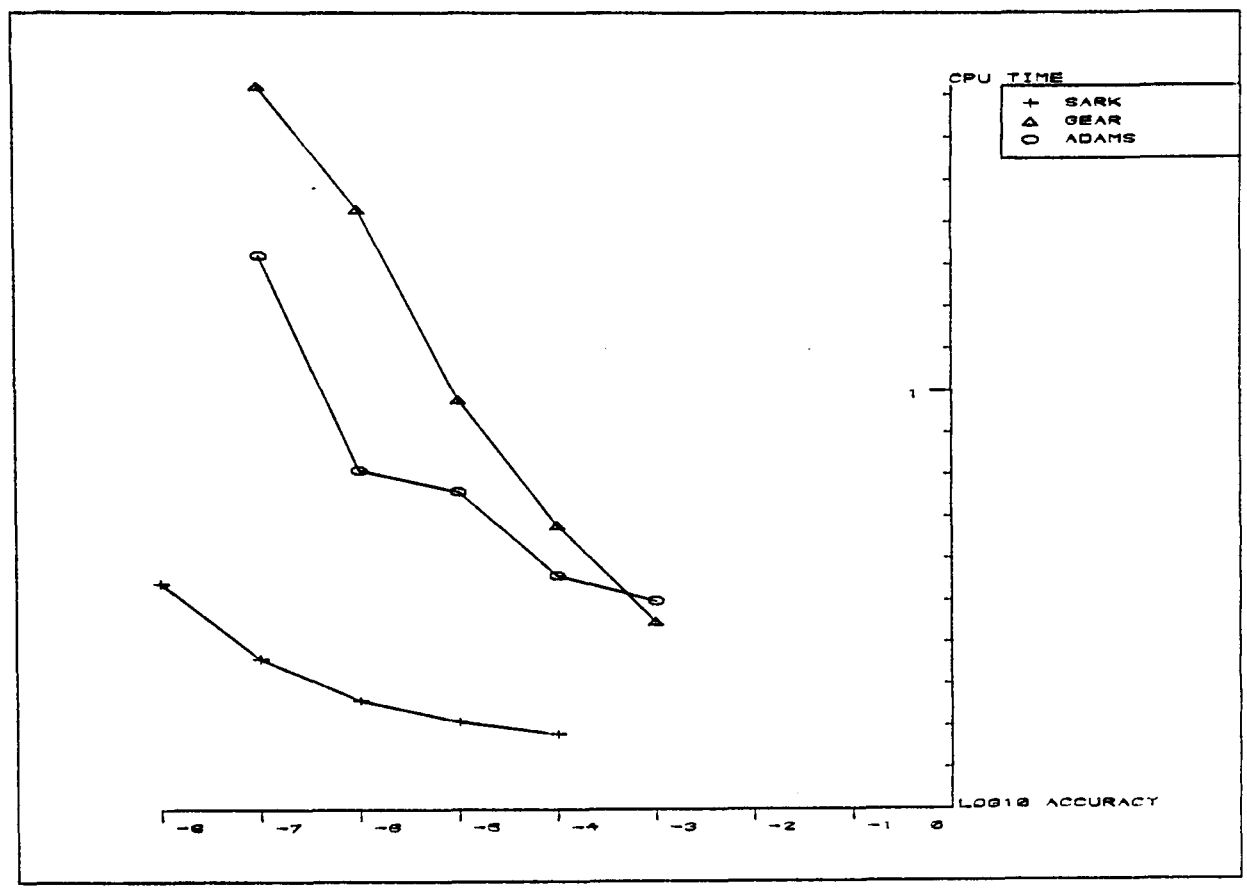


Figure 7.7 : Non-stiff problem B2

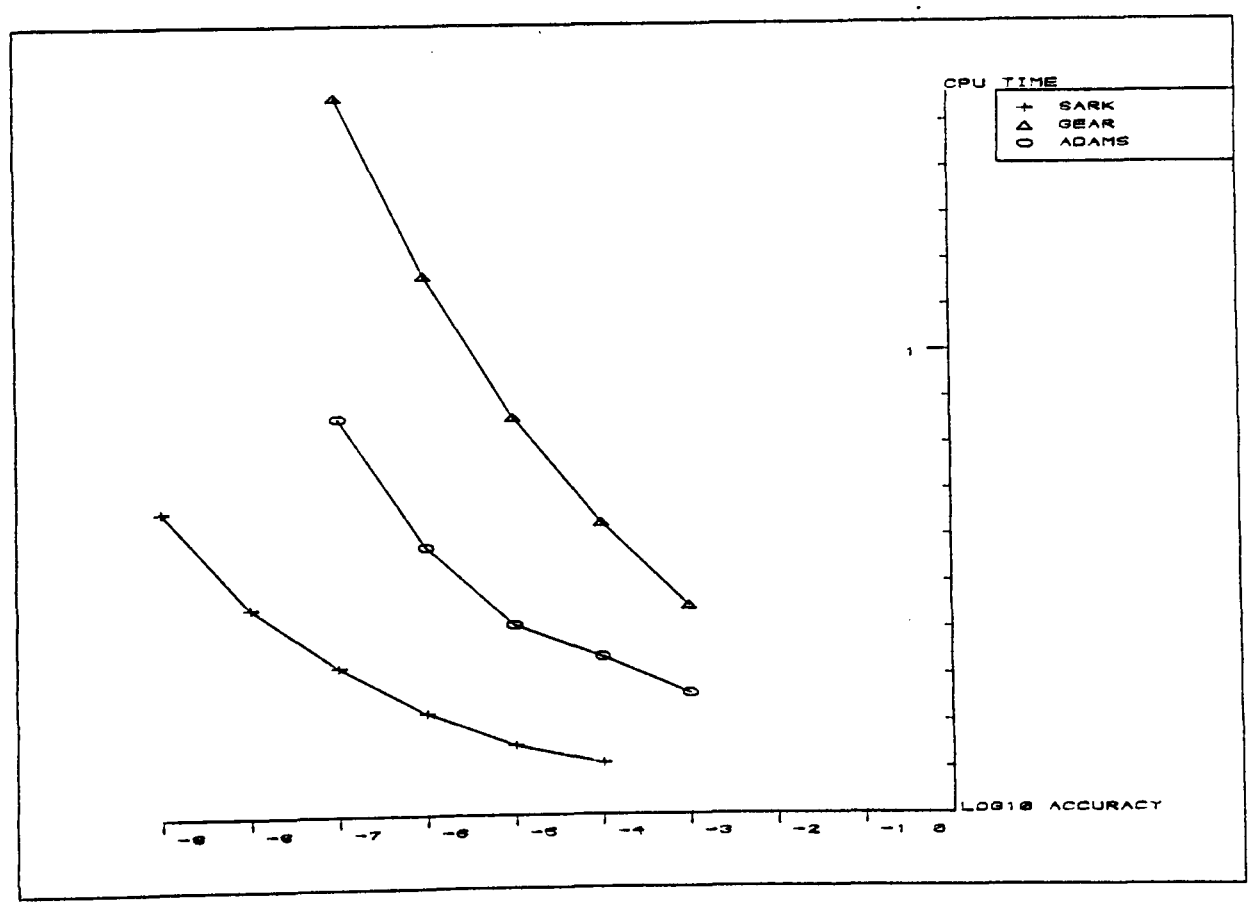


Figure 7.8 : Non-stiff problem B3

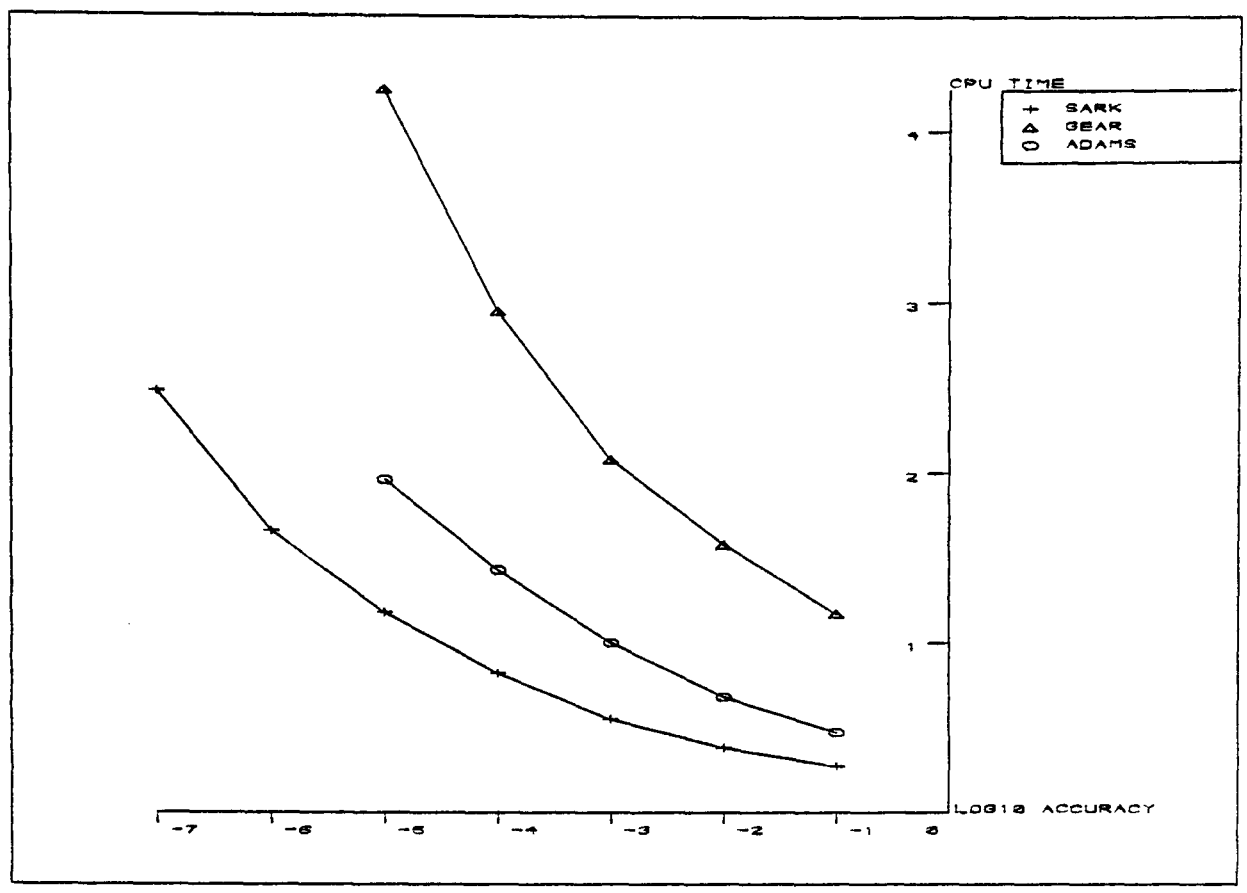


Figure 7.9 : Non-stiff problem B4

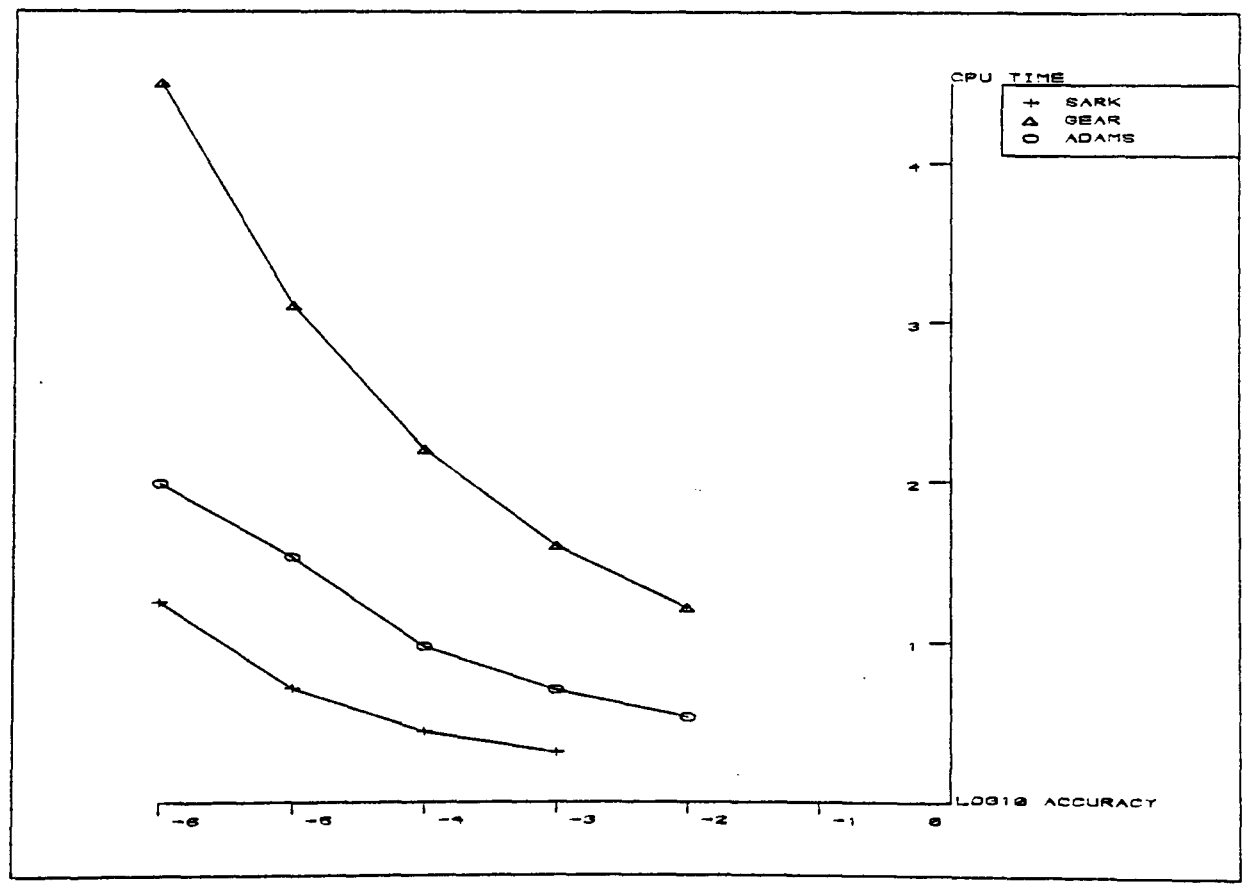


Figure 7.10 : Non-stiff problem B5

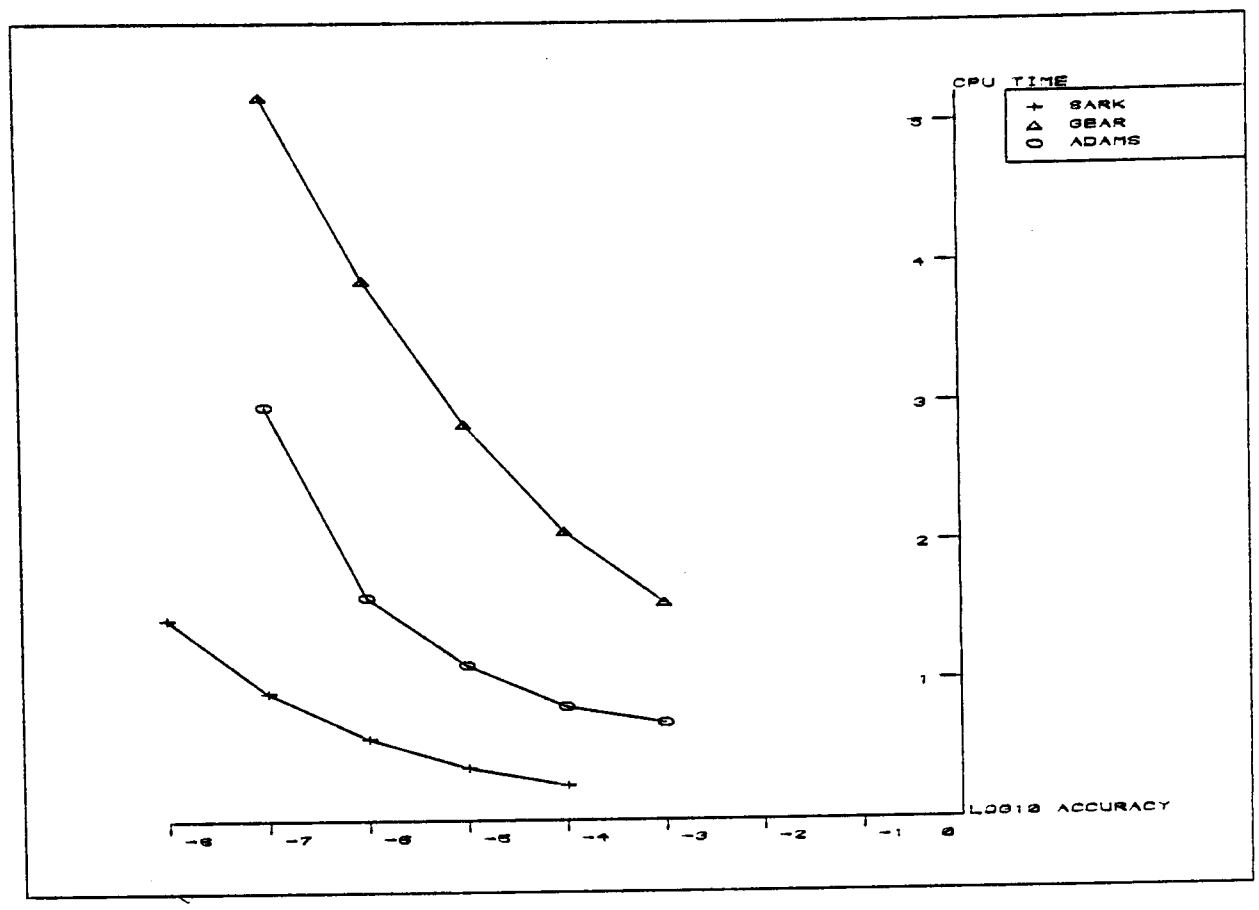


Figure 7.11 : Non-stiff problem C1

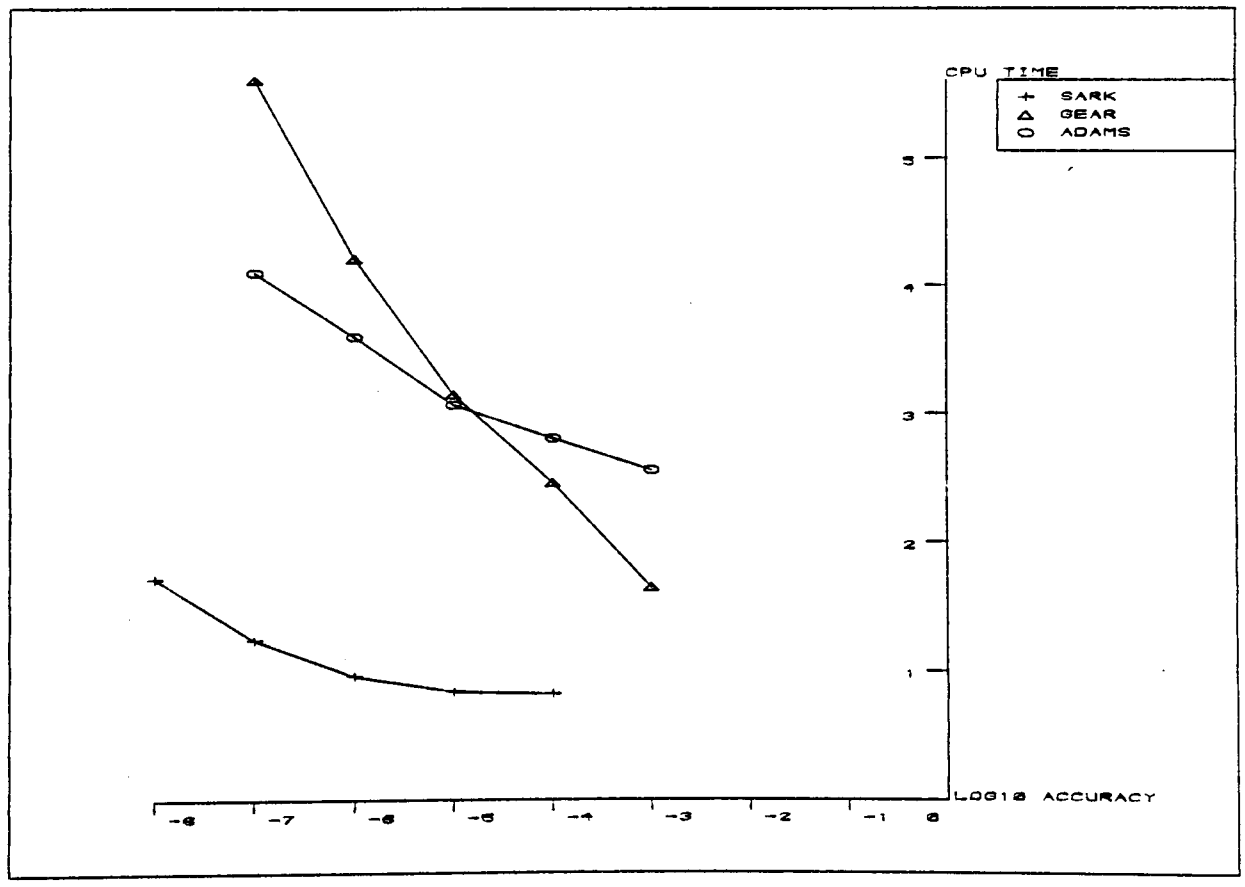


Figure 7.12 : Non-stiff problem C2

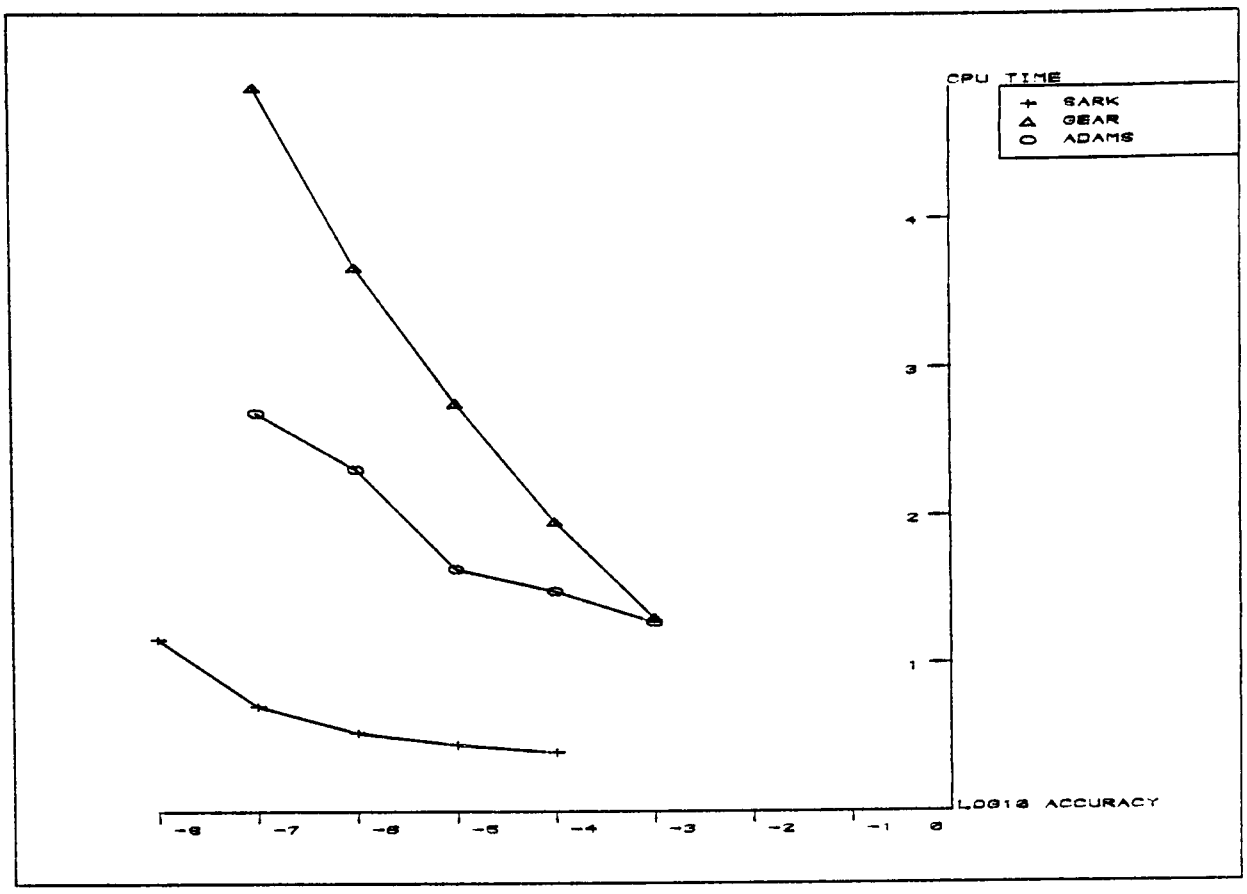


Figure 7.13 : Non-stiff problem C3

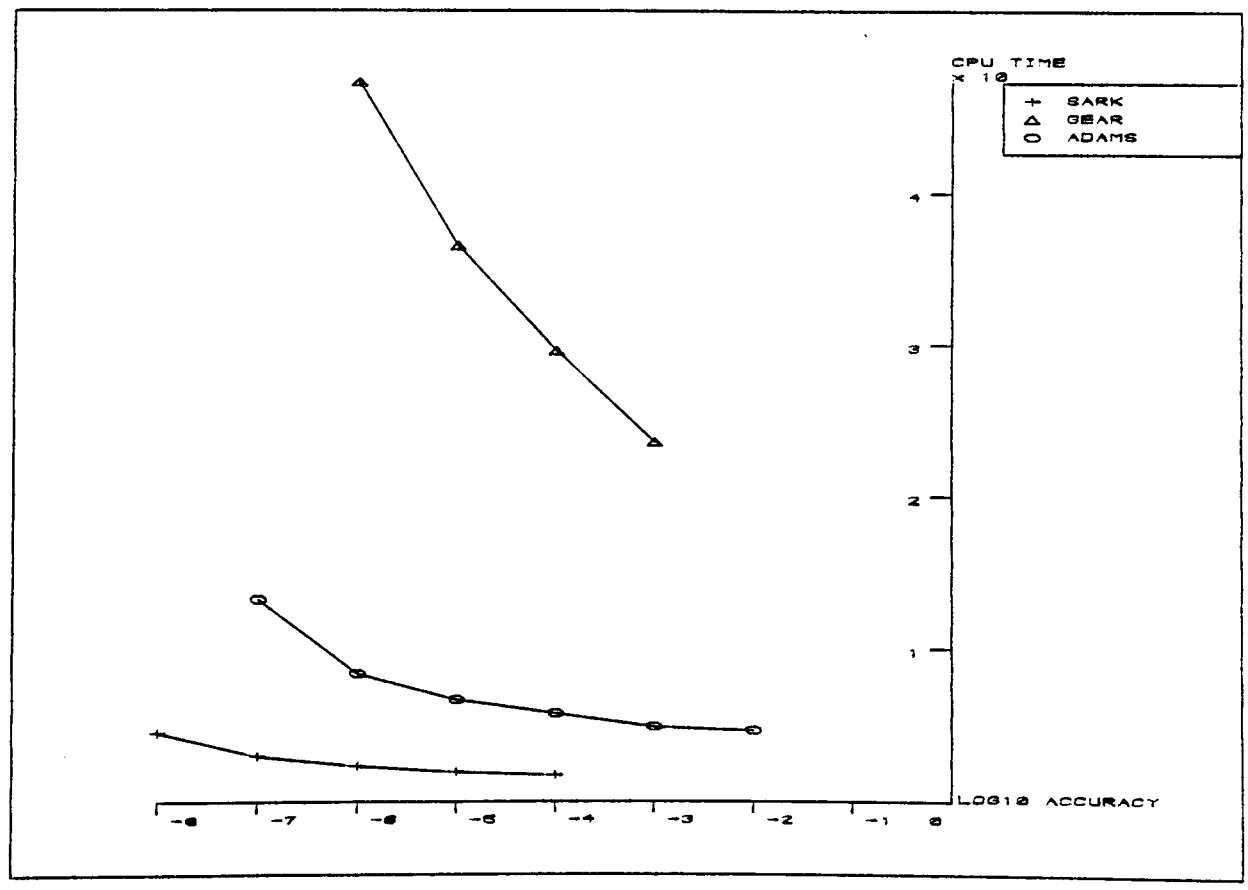


Figure 7.14 : Non-stiff problem C4

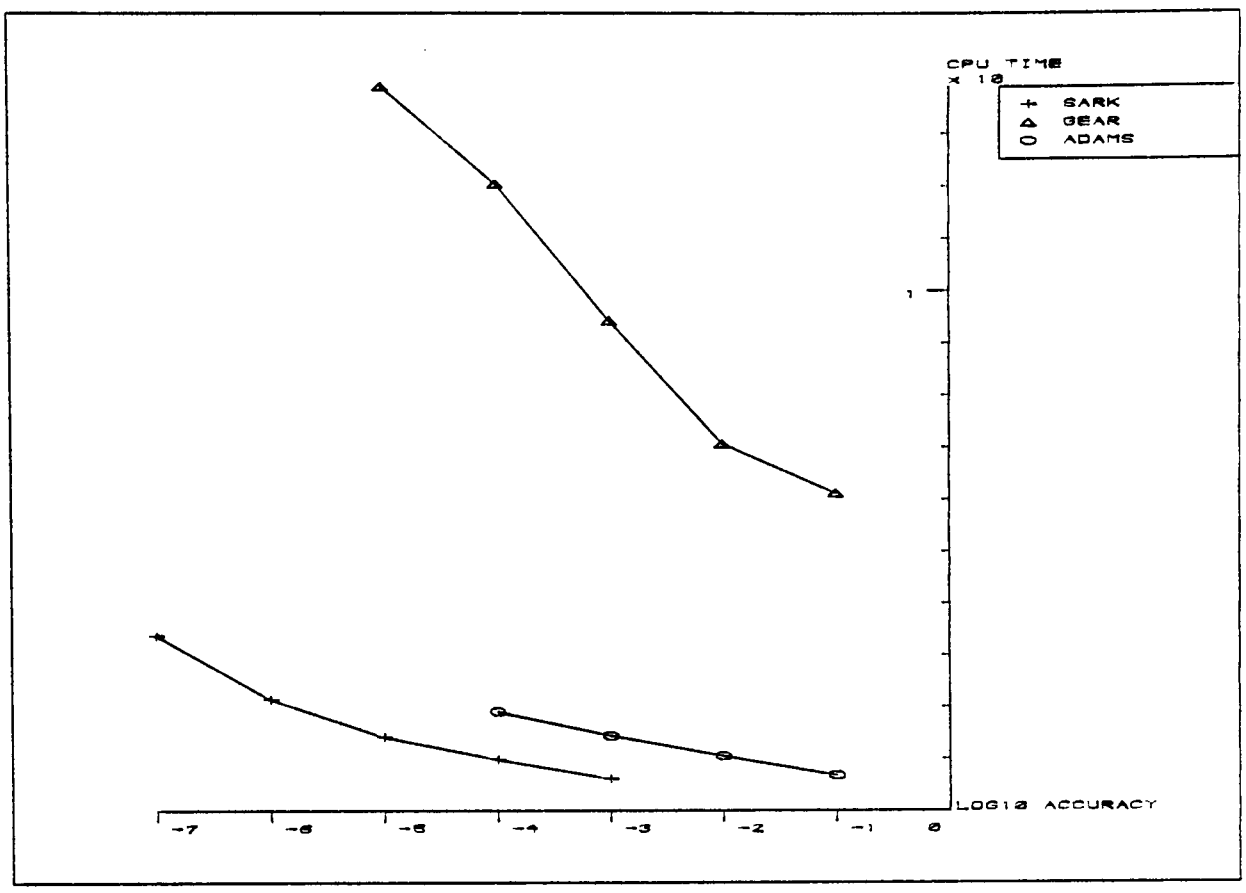


Figure 7.15 : Non-stiff problem C5

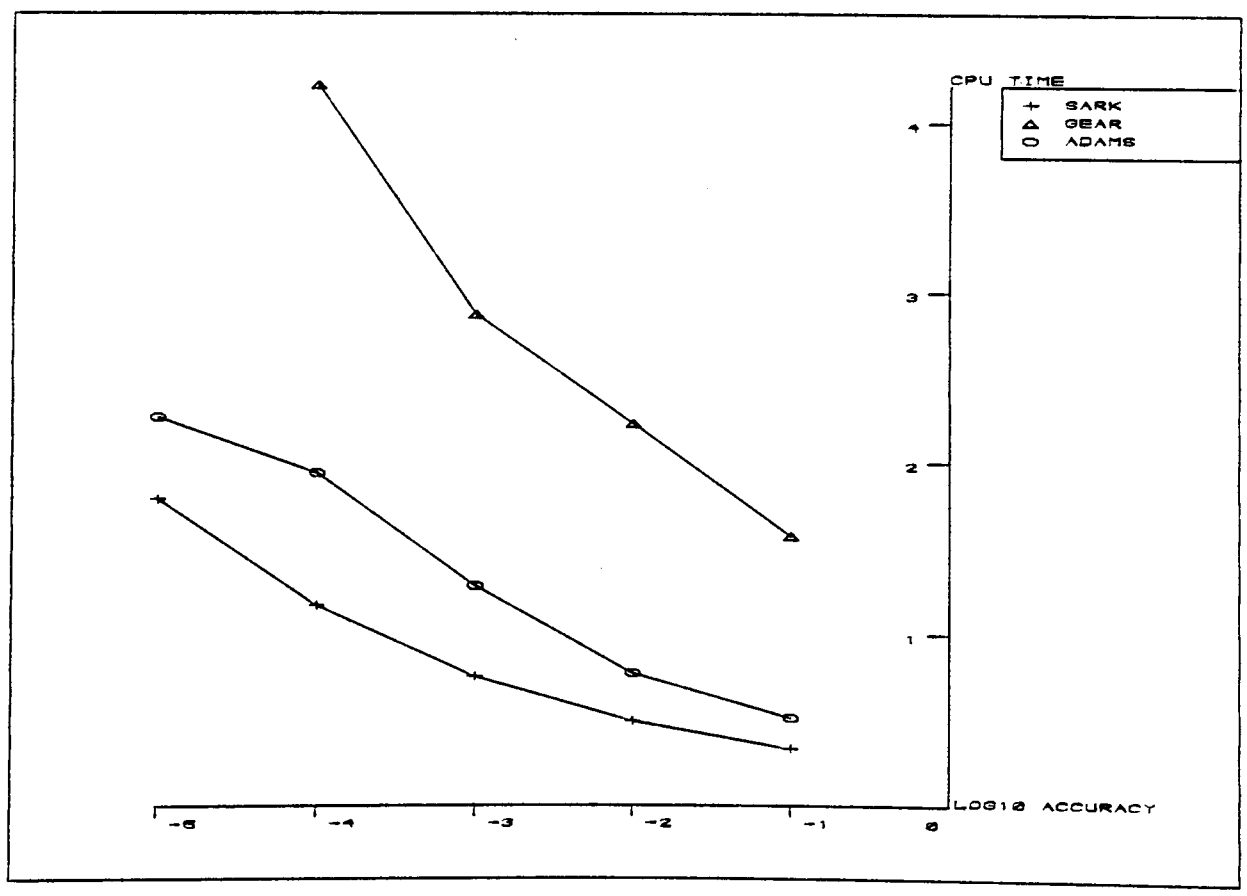


Figure 7.16 : Non-stiff problem D1

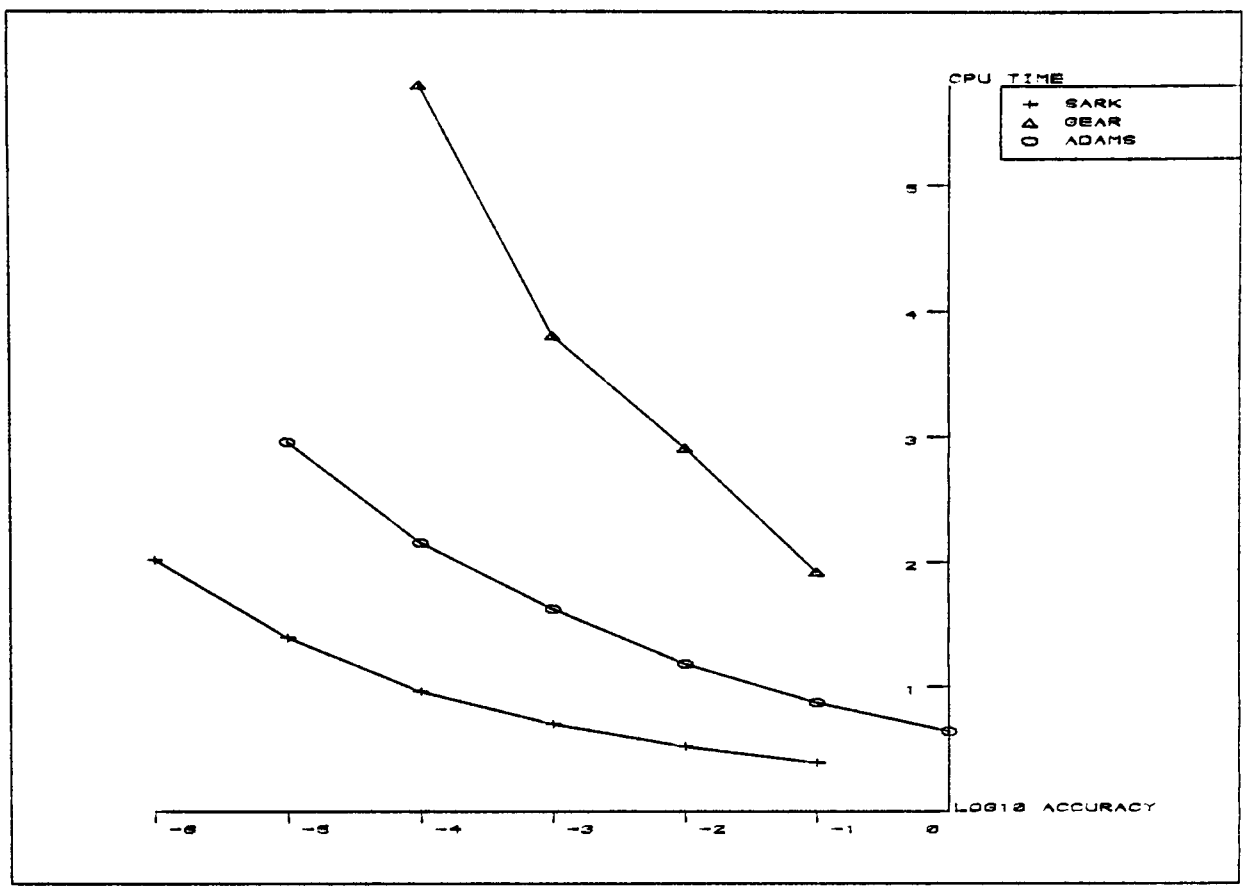


Figure 7.17 : Non-stiff problem D2

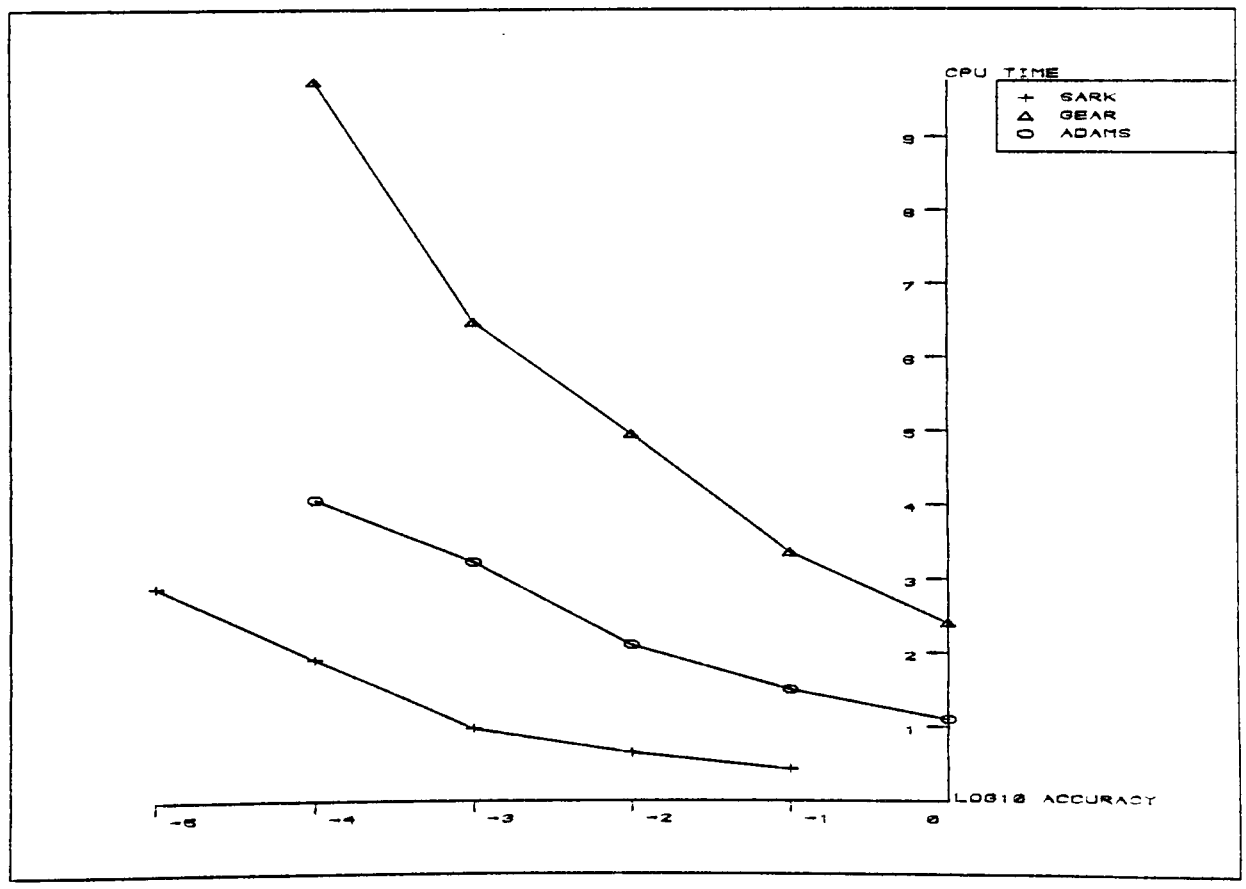


Figure 7.18 : Non-stiff problem D3

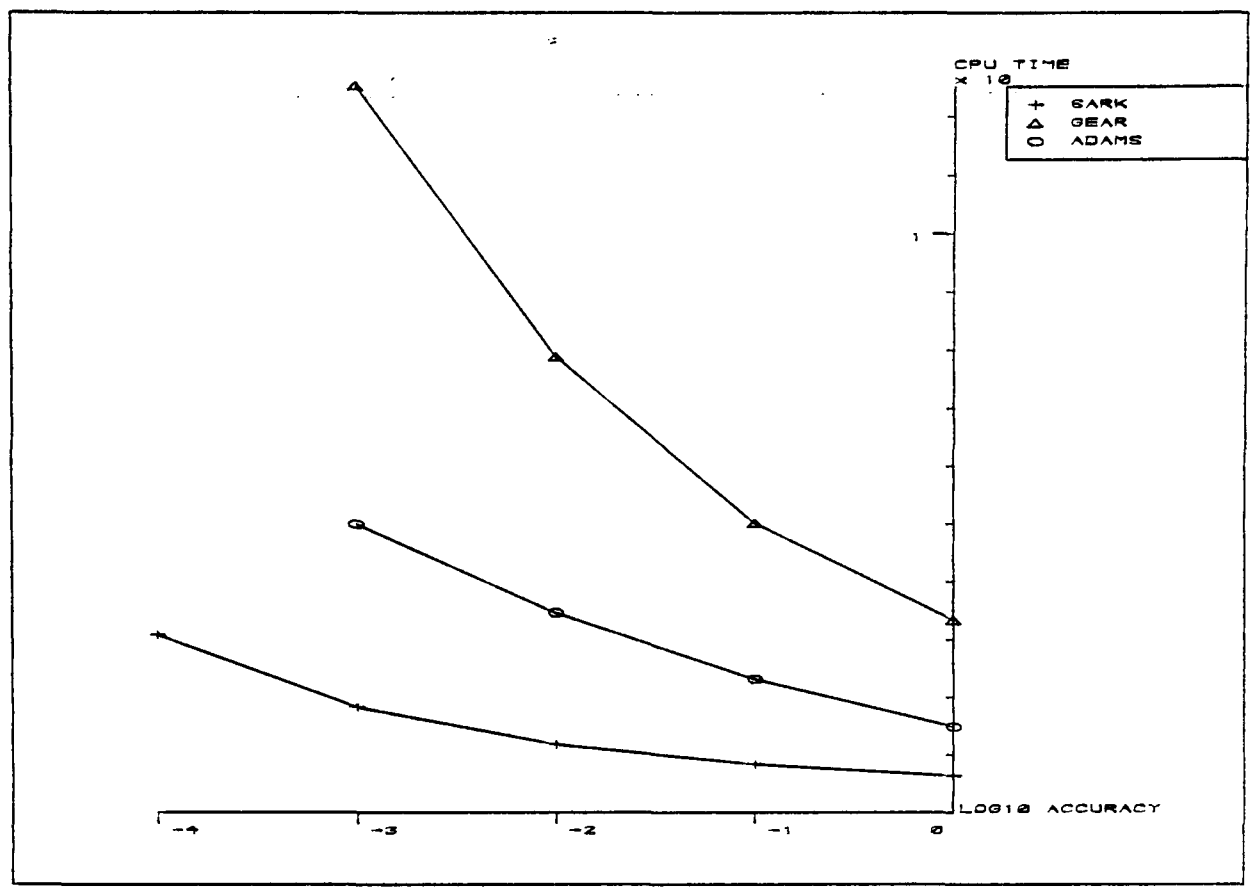


Figure 7.19 : Non-stiff problem D4

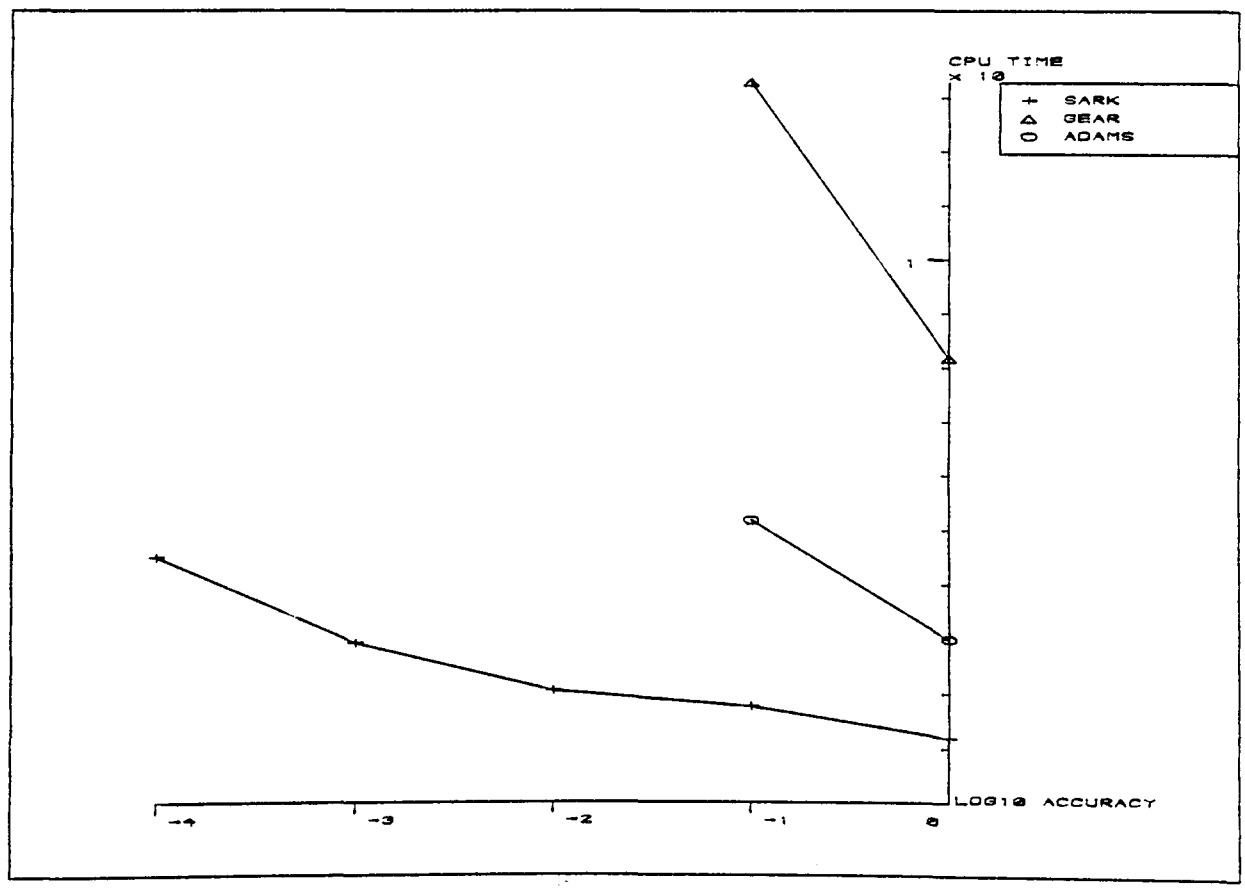


Figure 7.20 : Non-stiff problem D5

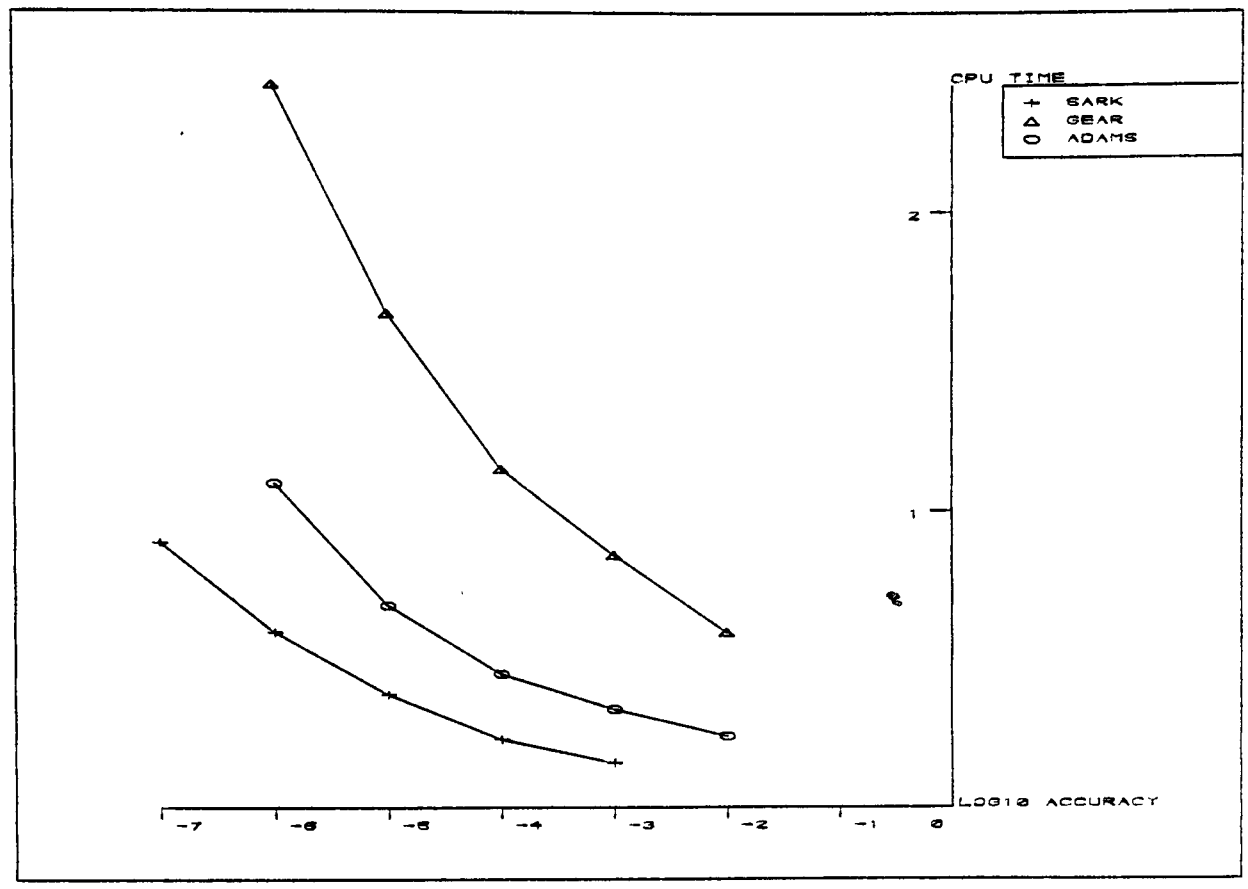


Figure 7.21 : Non-stiff problem E1

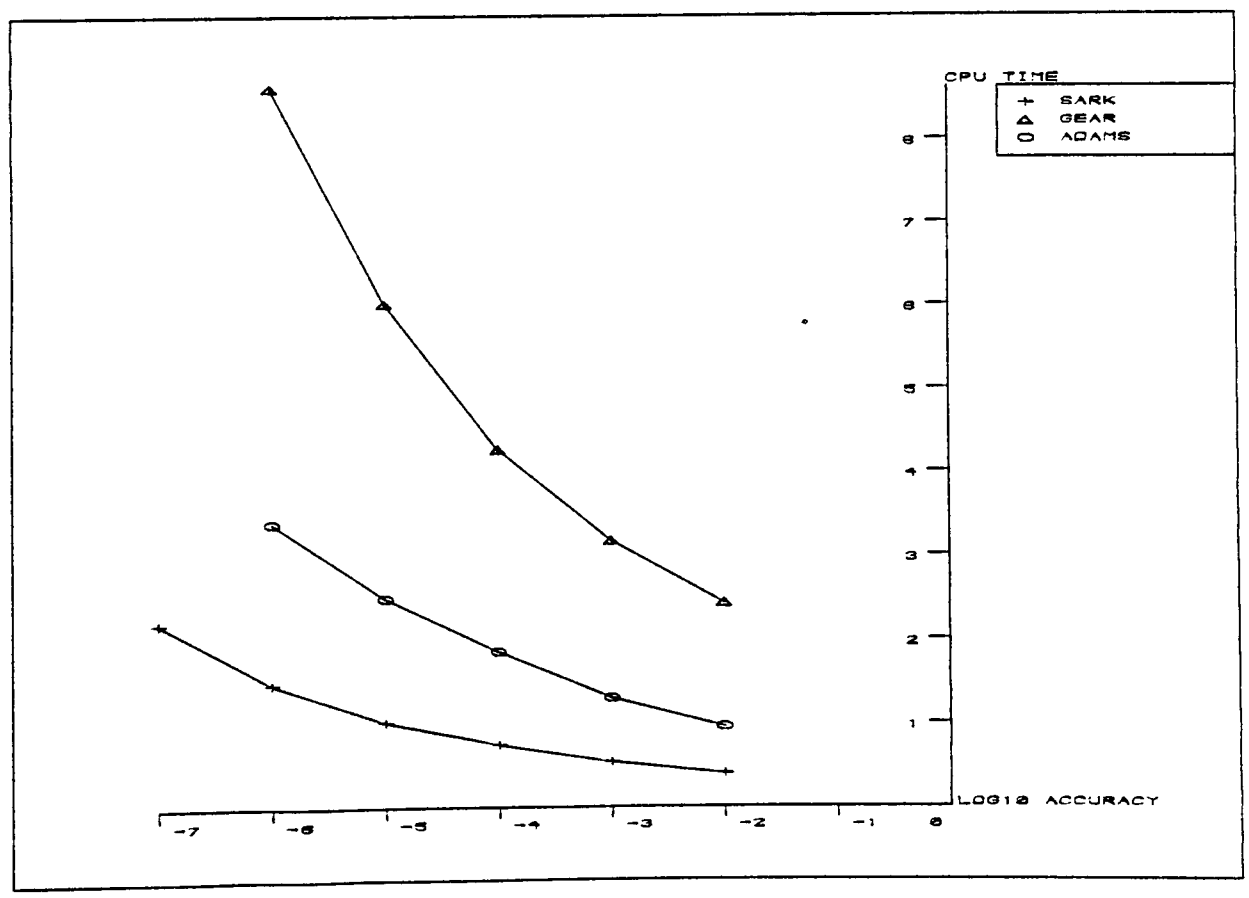


Figure 7.22 : Non-stiff problem E2

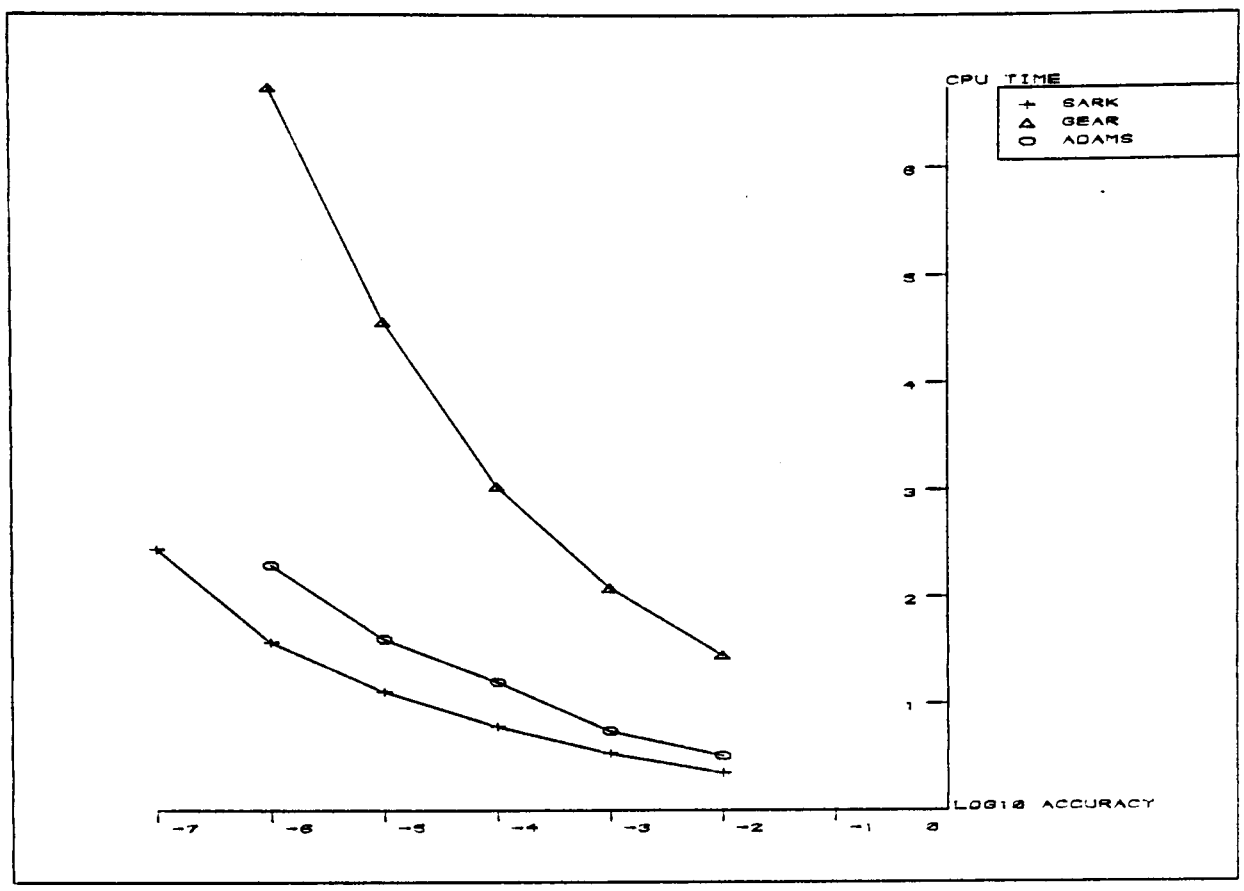


Figure 7.23 : Non-stiff problem E3

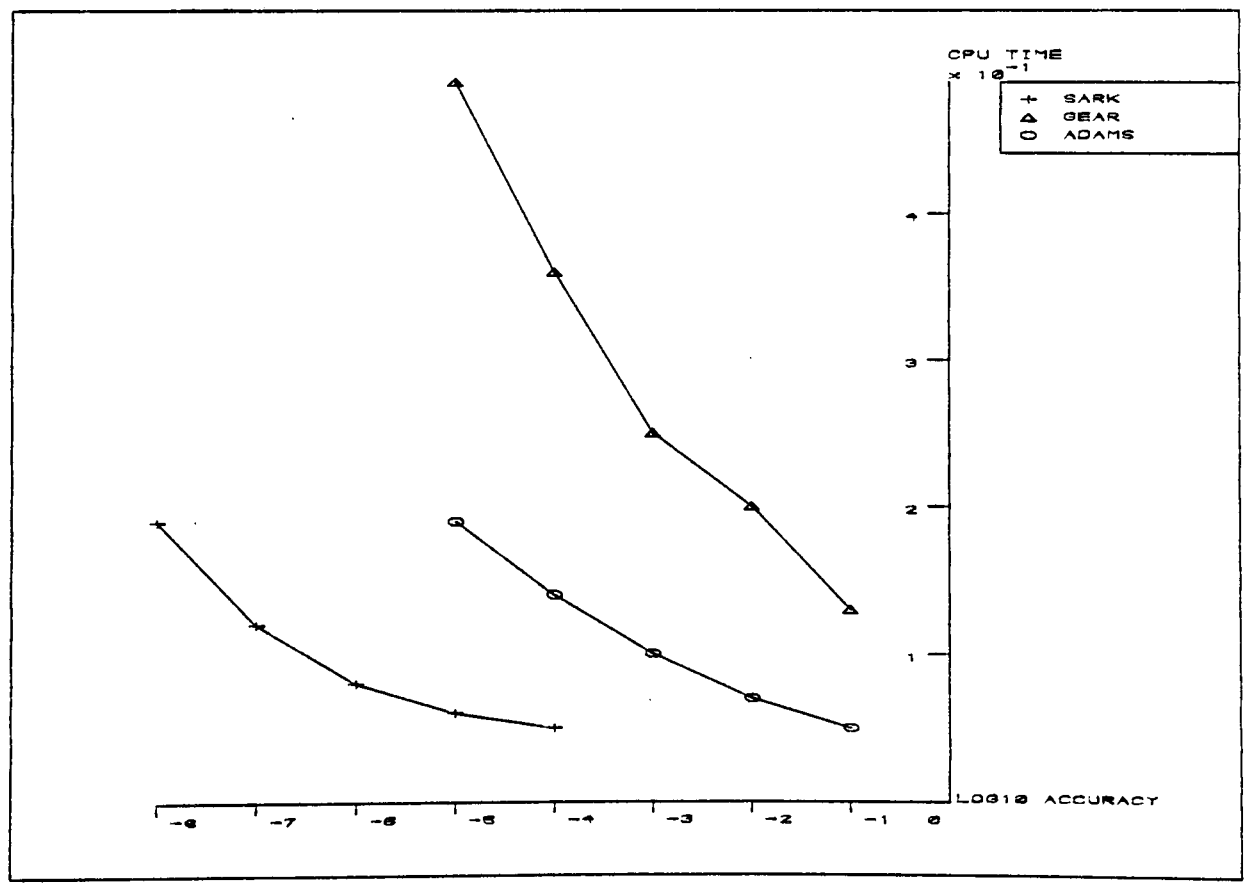


Figure 7.24 : Non-stiff problem E4

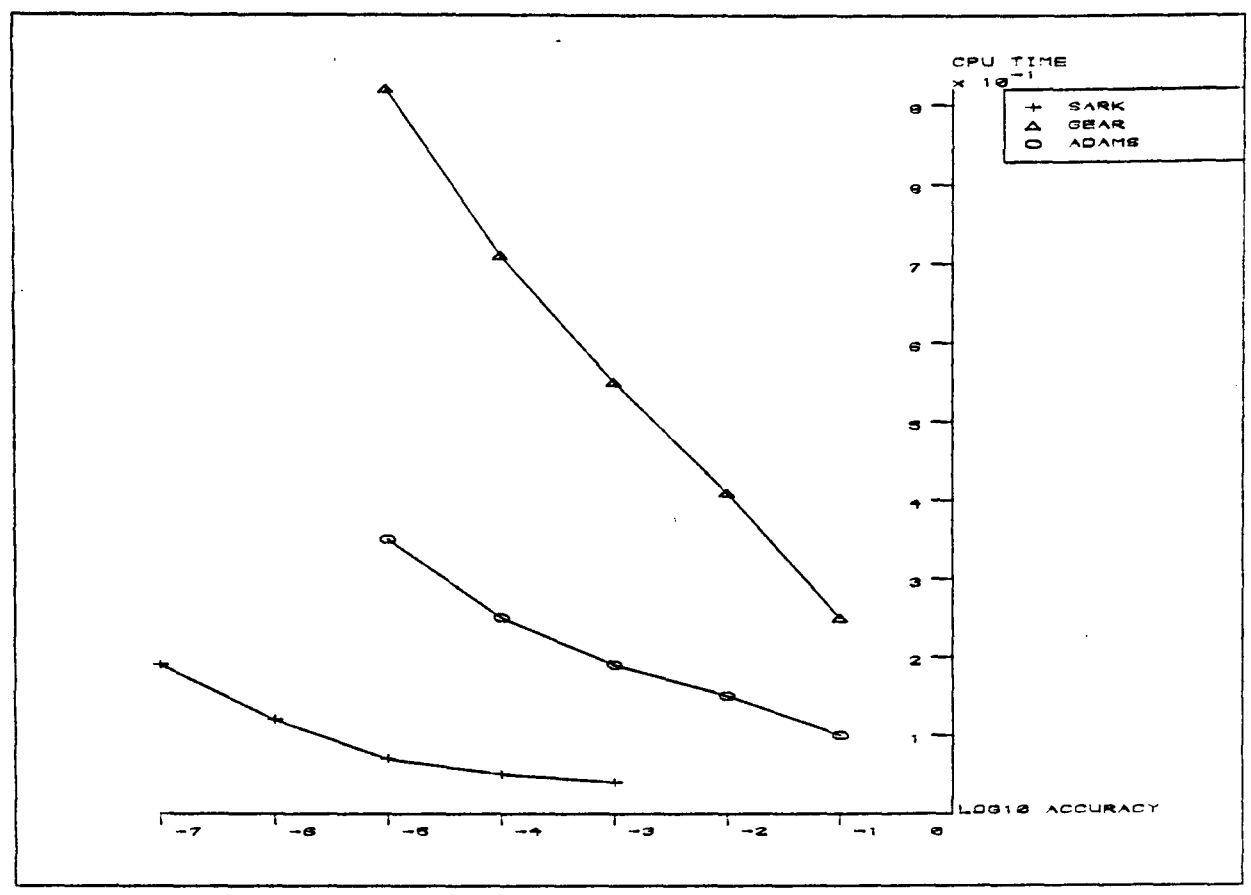


Figure 7.25 : Non-stiff problem E5

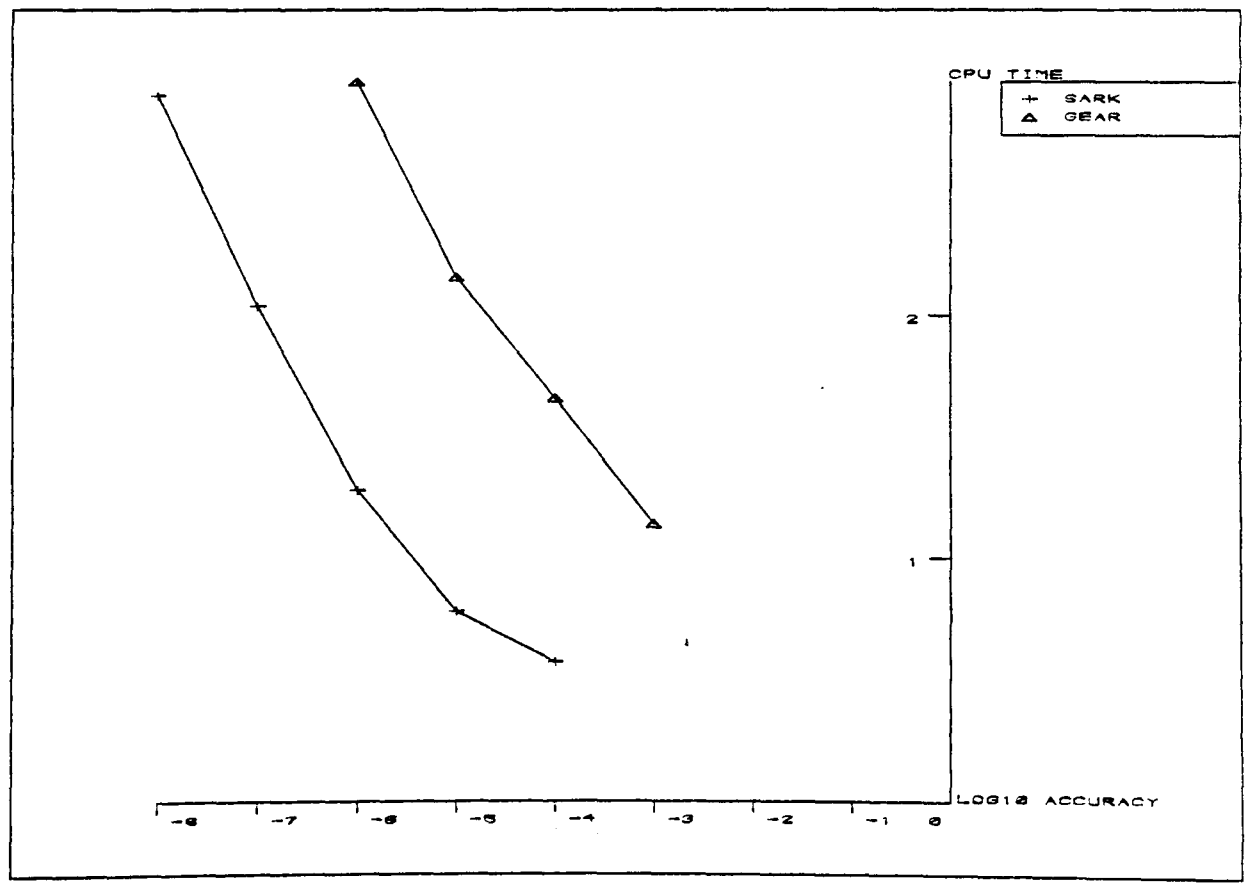


Figure 7.26 : Stiff problem A1

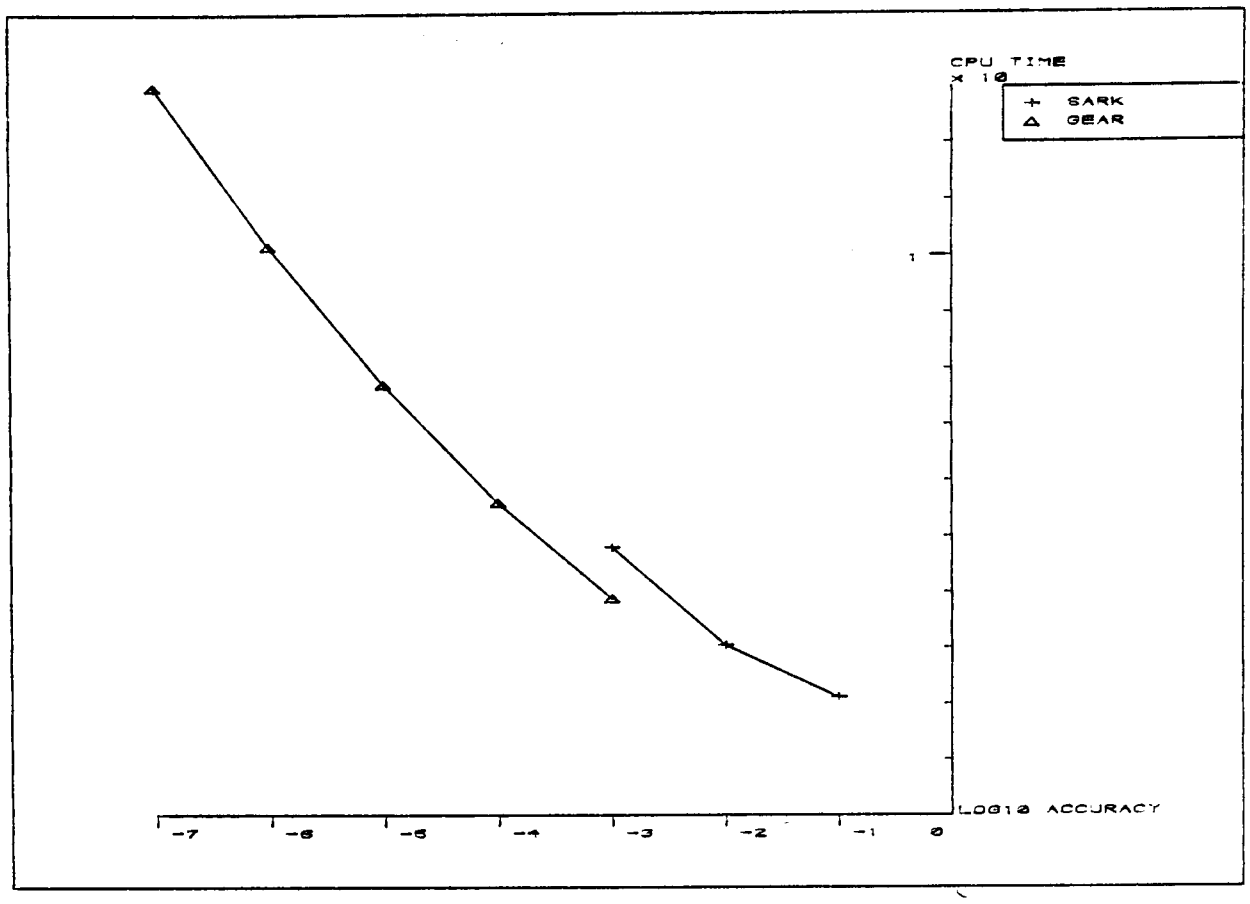


Figure 7.27 : Stiff problem A2

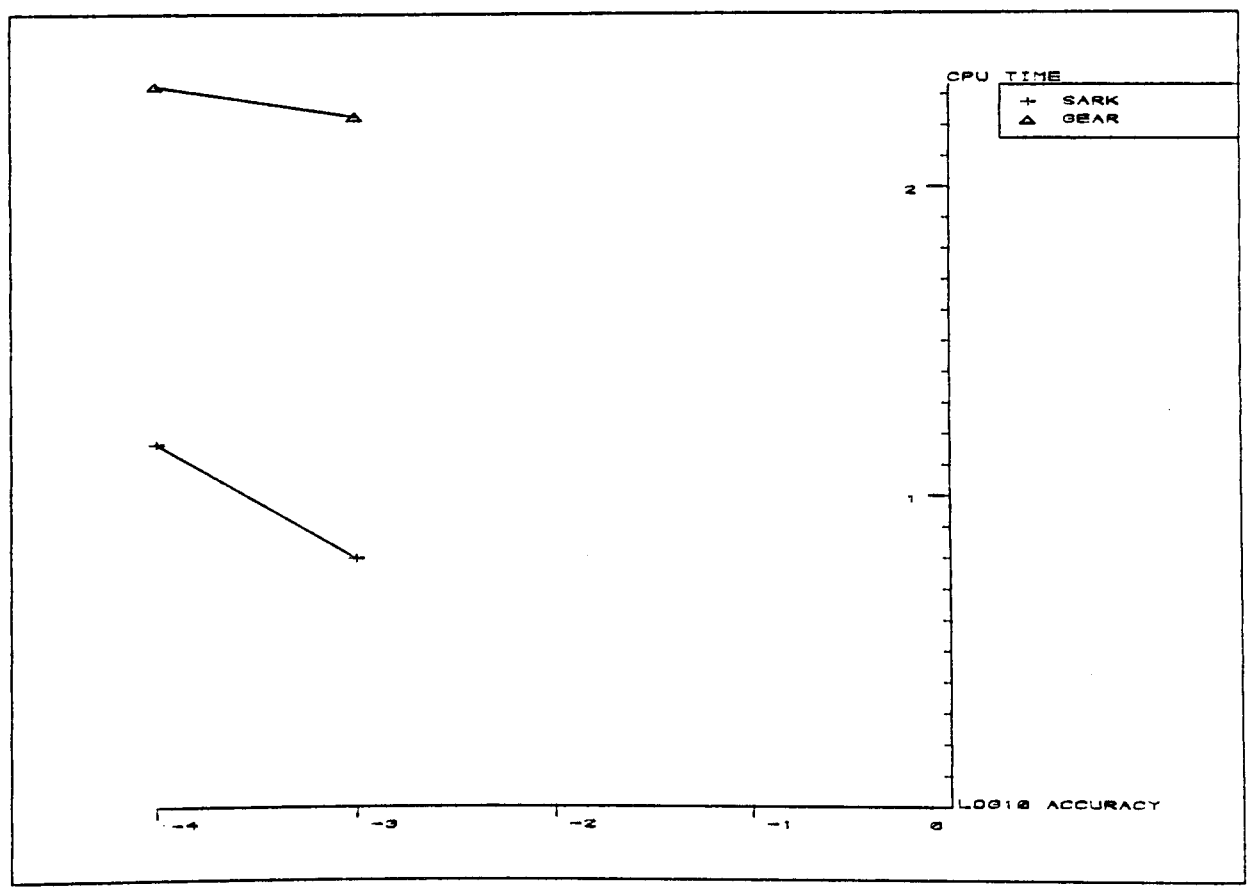


Figure 7.28 : Stiff problem A3

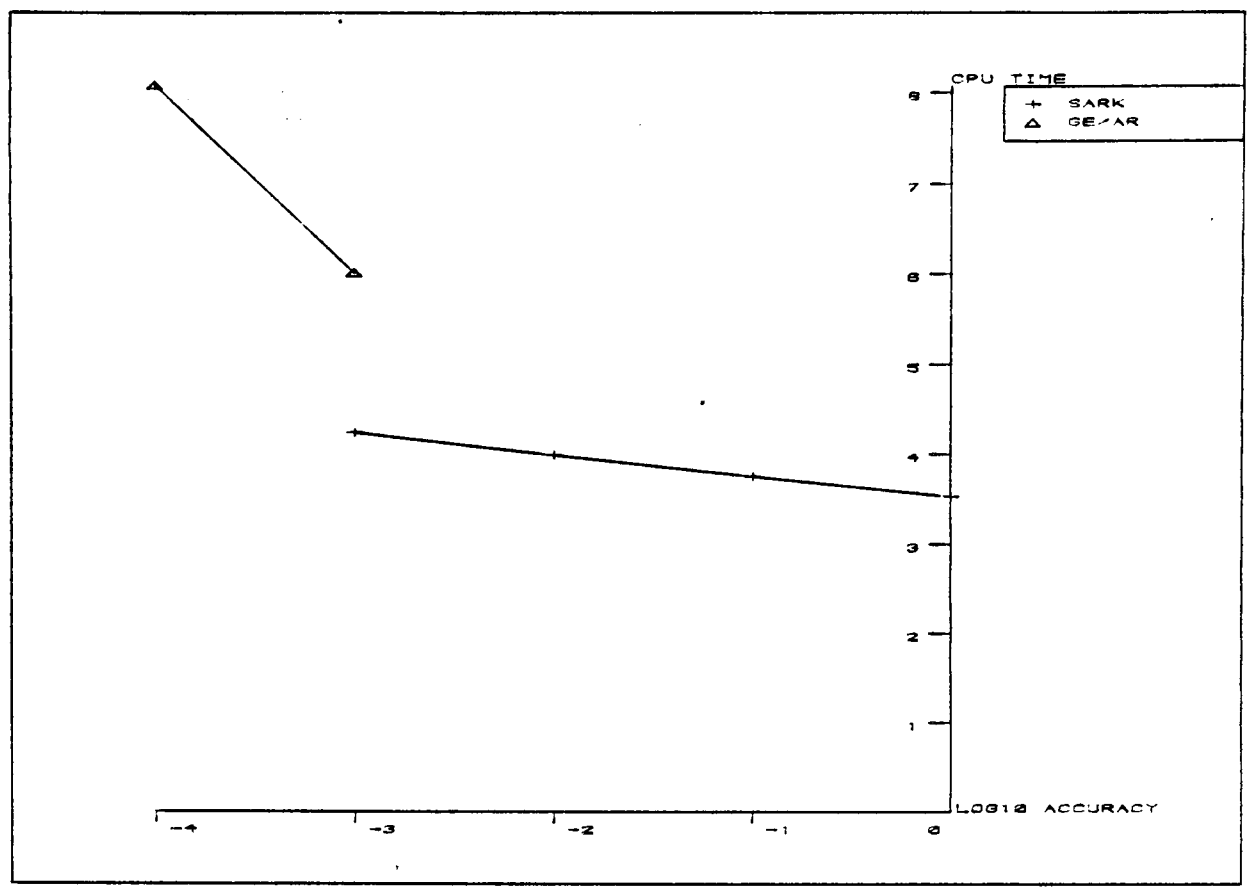


Figure 7.29 : Stiff problem A4

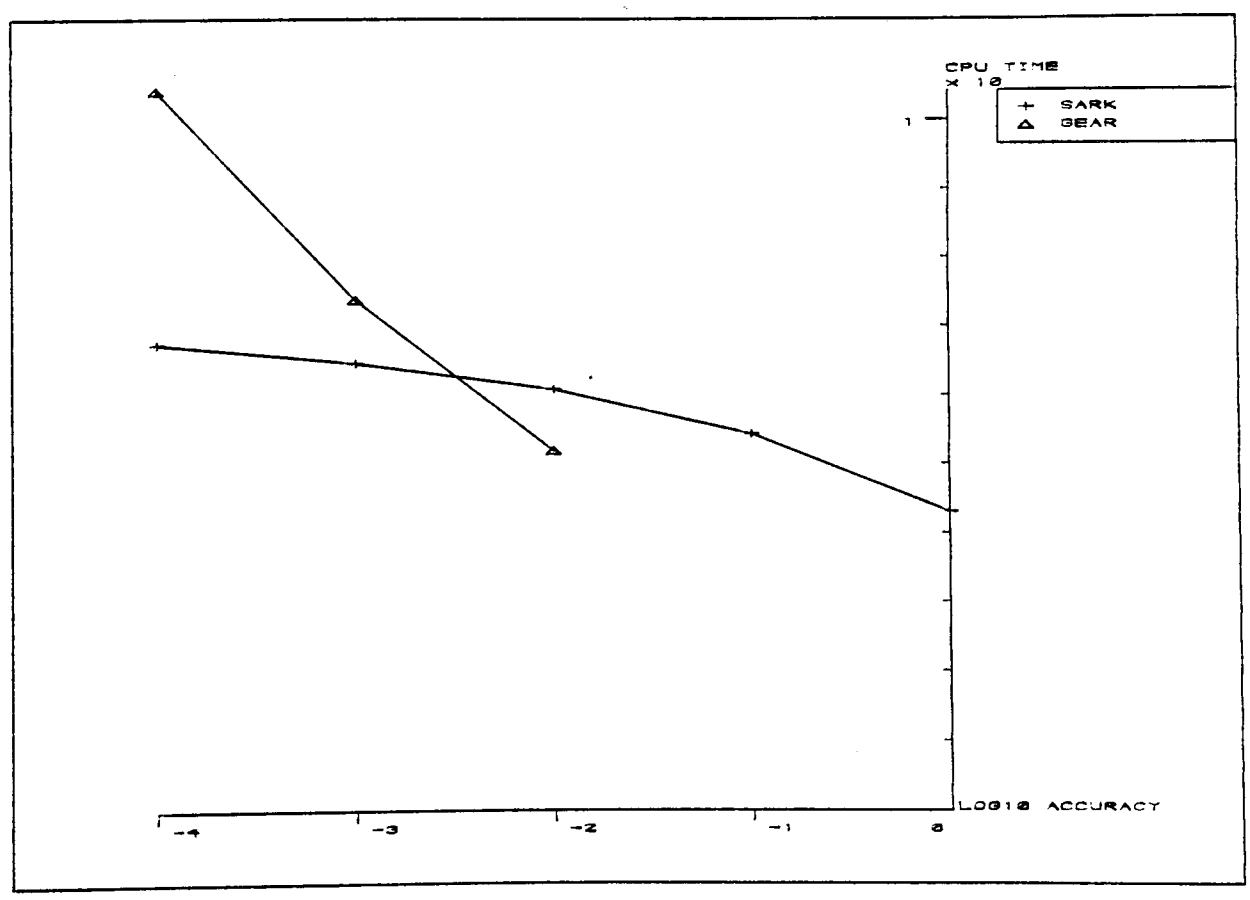


Figure 7.30 : Stiff problem B1

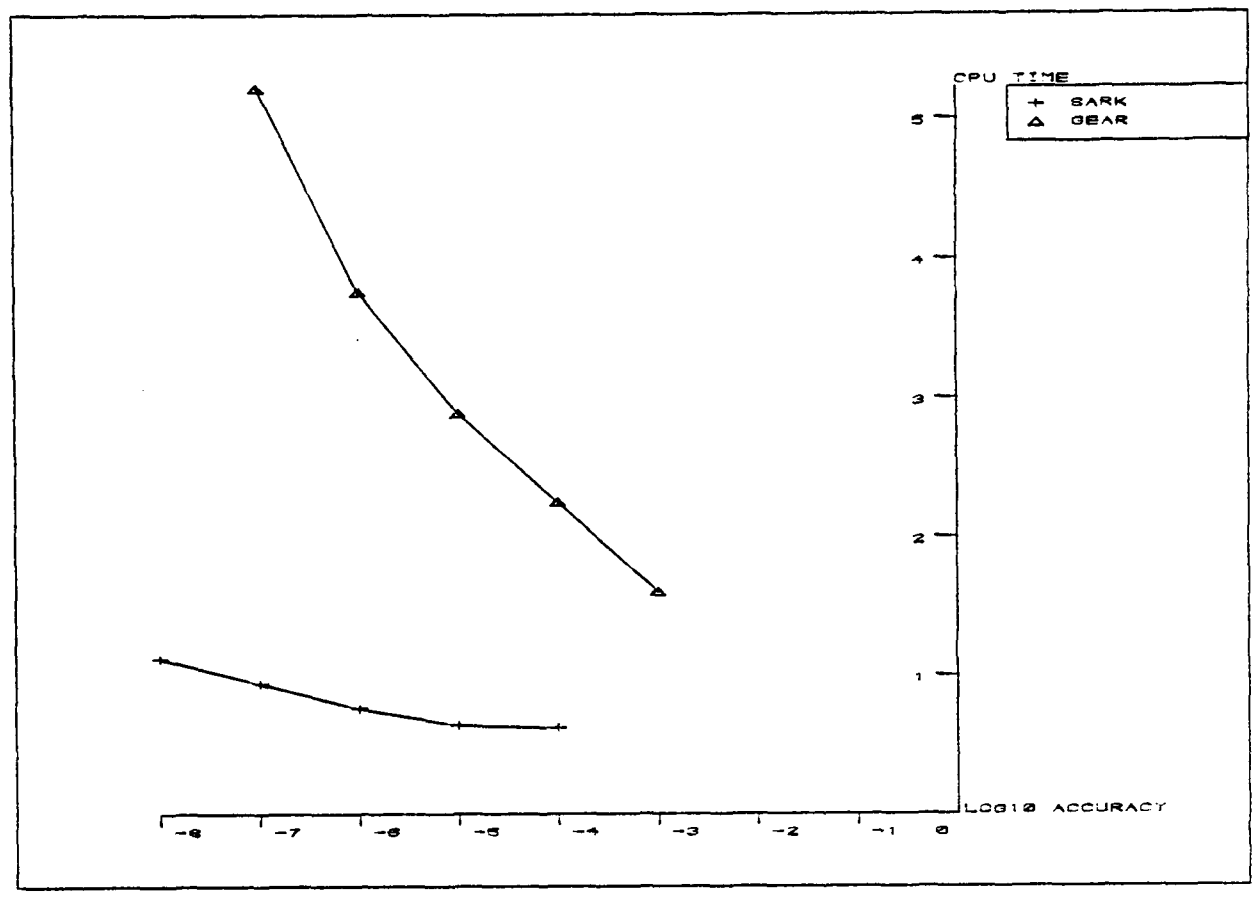


Figure 7.31 : Stiff problem B2

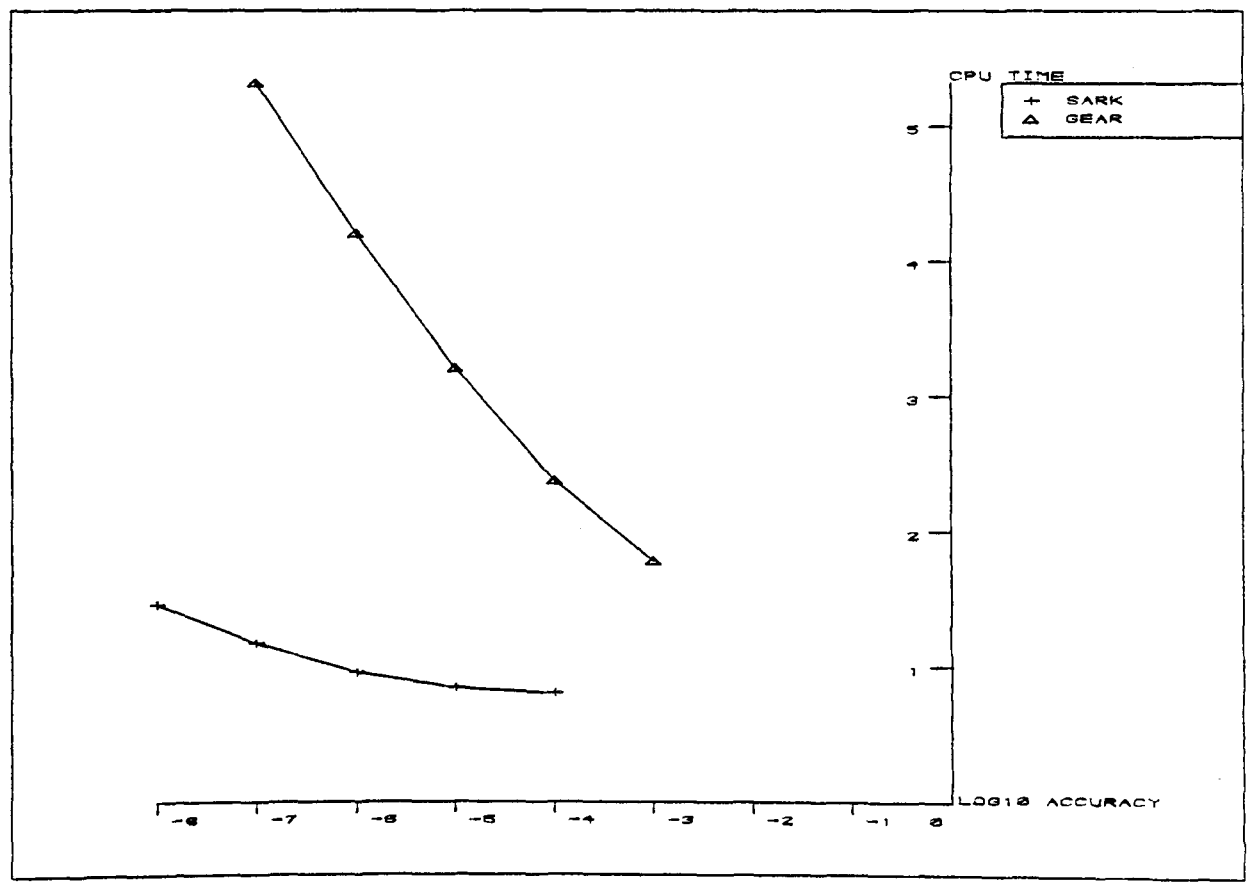


Figure 7.32 : Stiff problem B3

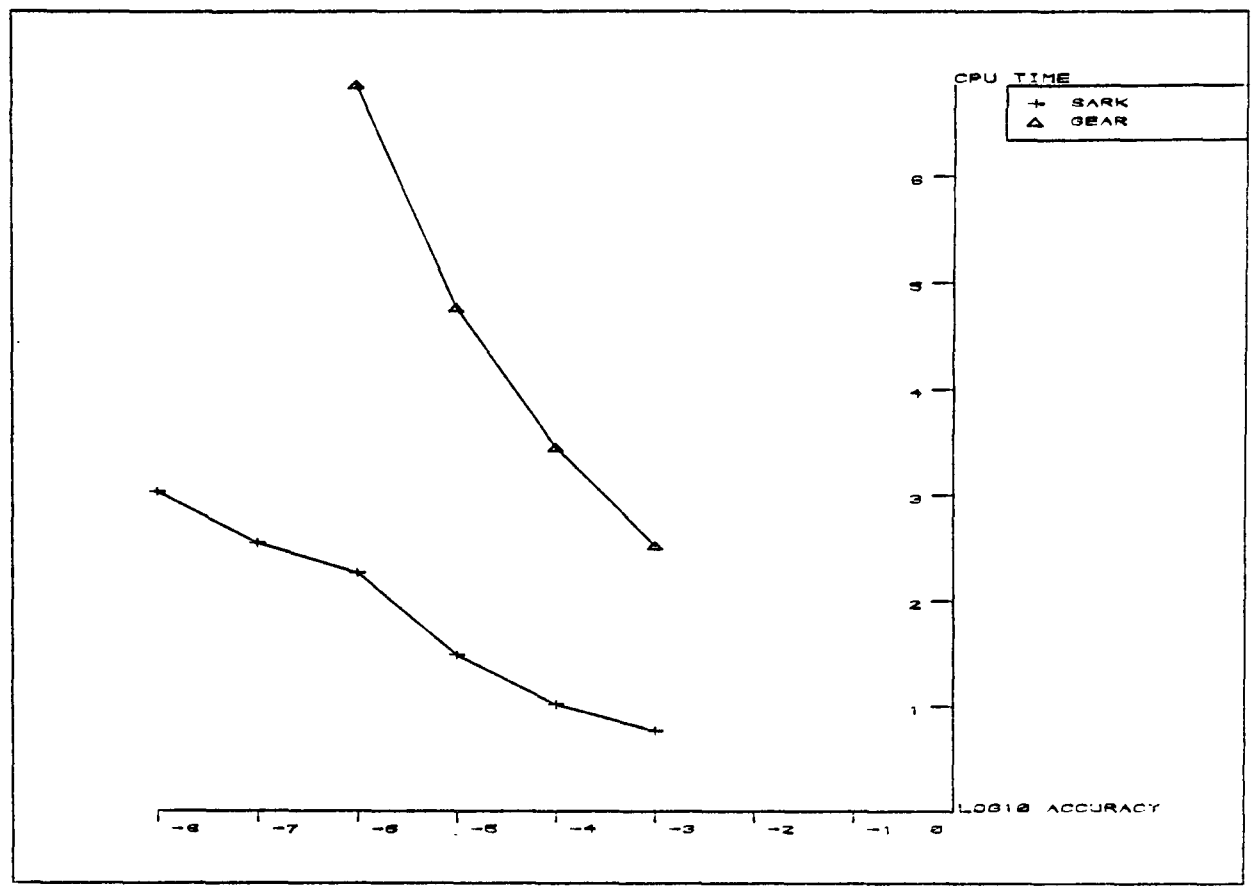


Figure 7.33 : Stiff problem B4

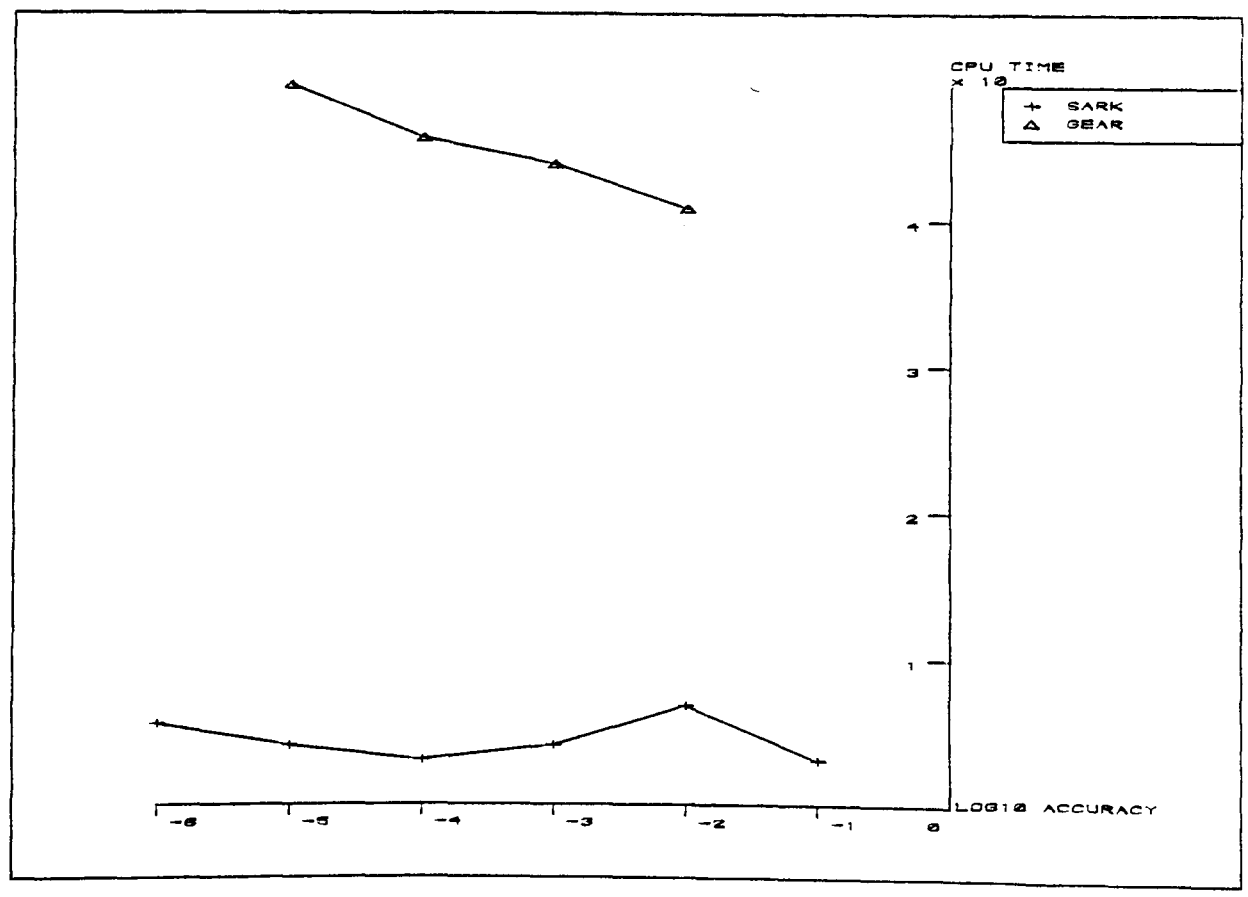


Figure 7.34 : Stiff problem B5

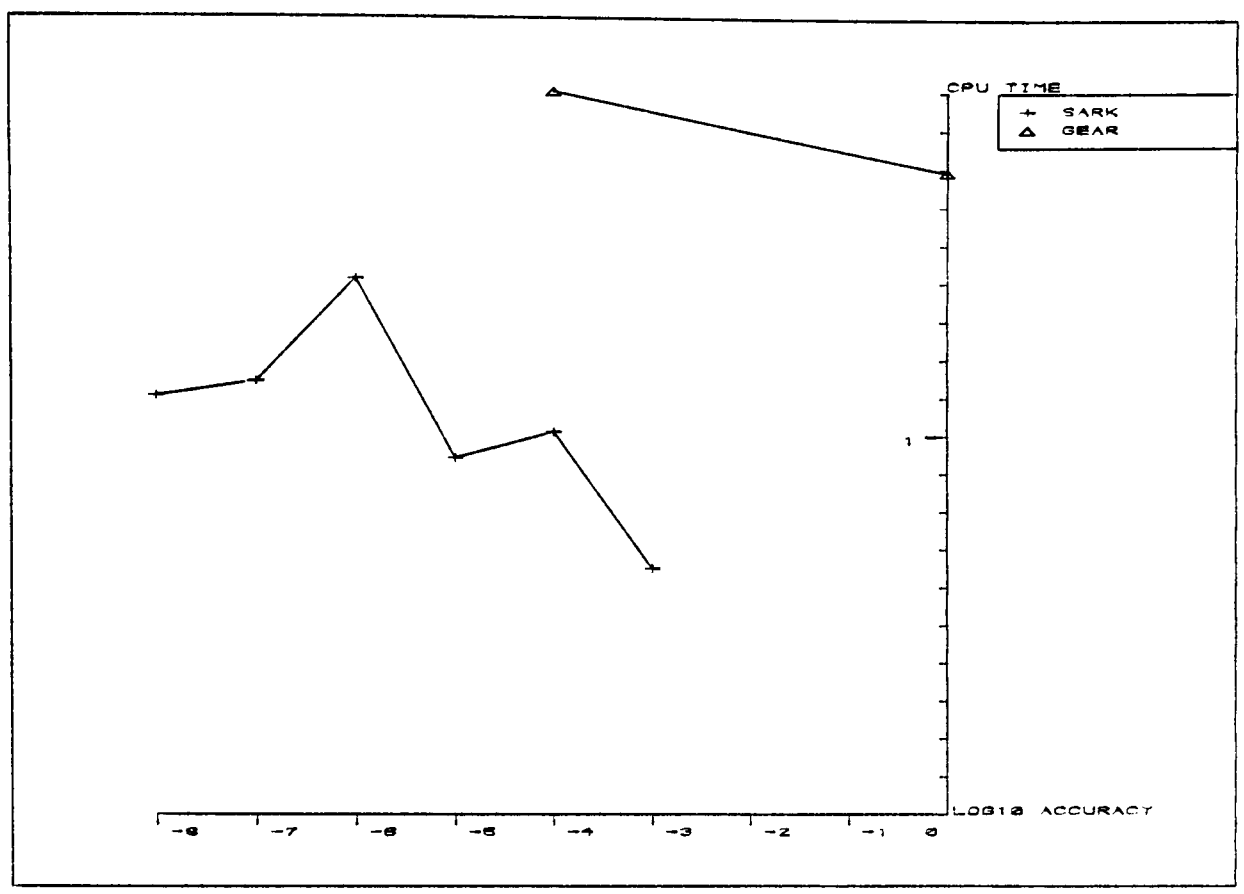


Figure 7.35 : Stiff problem C1

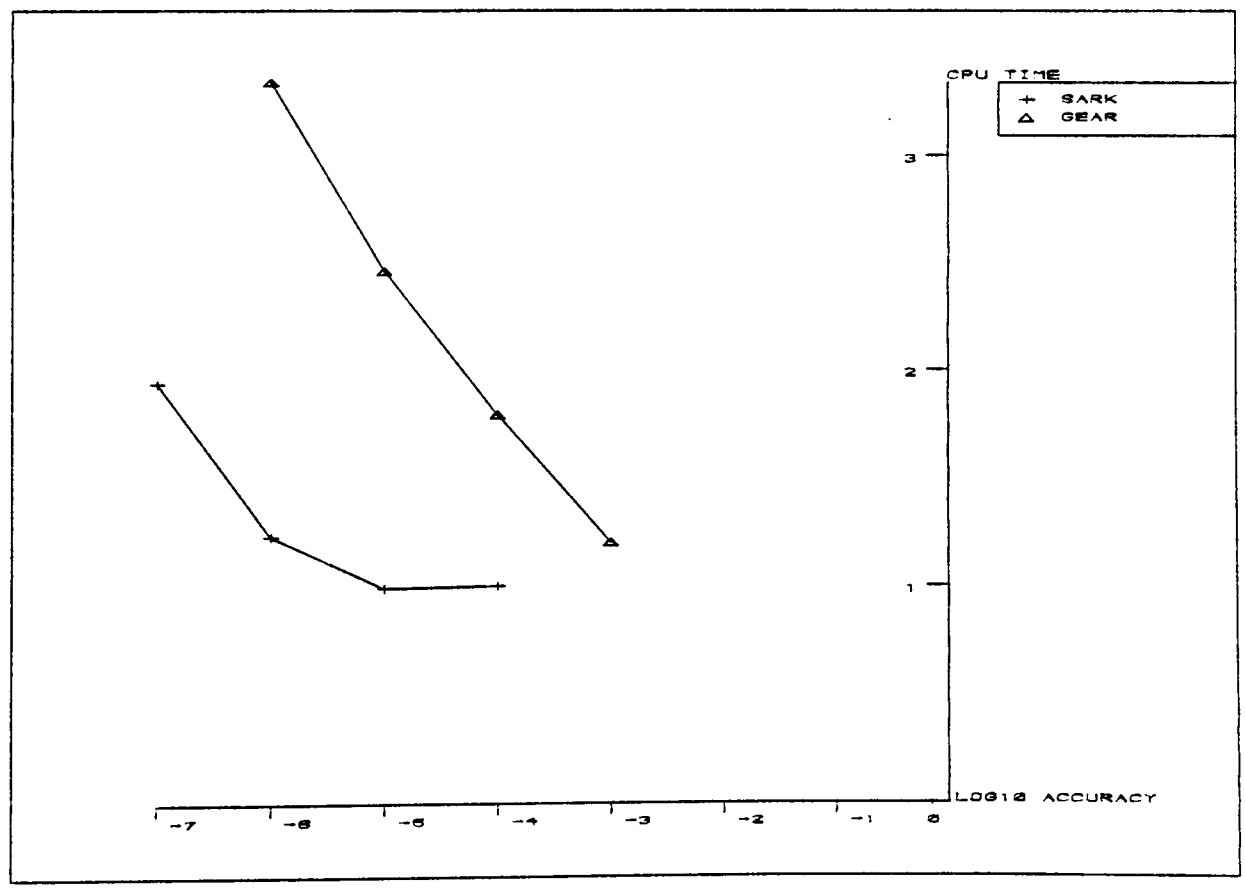


Figure 7.36 : Stiff problem C2

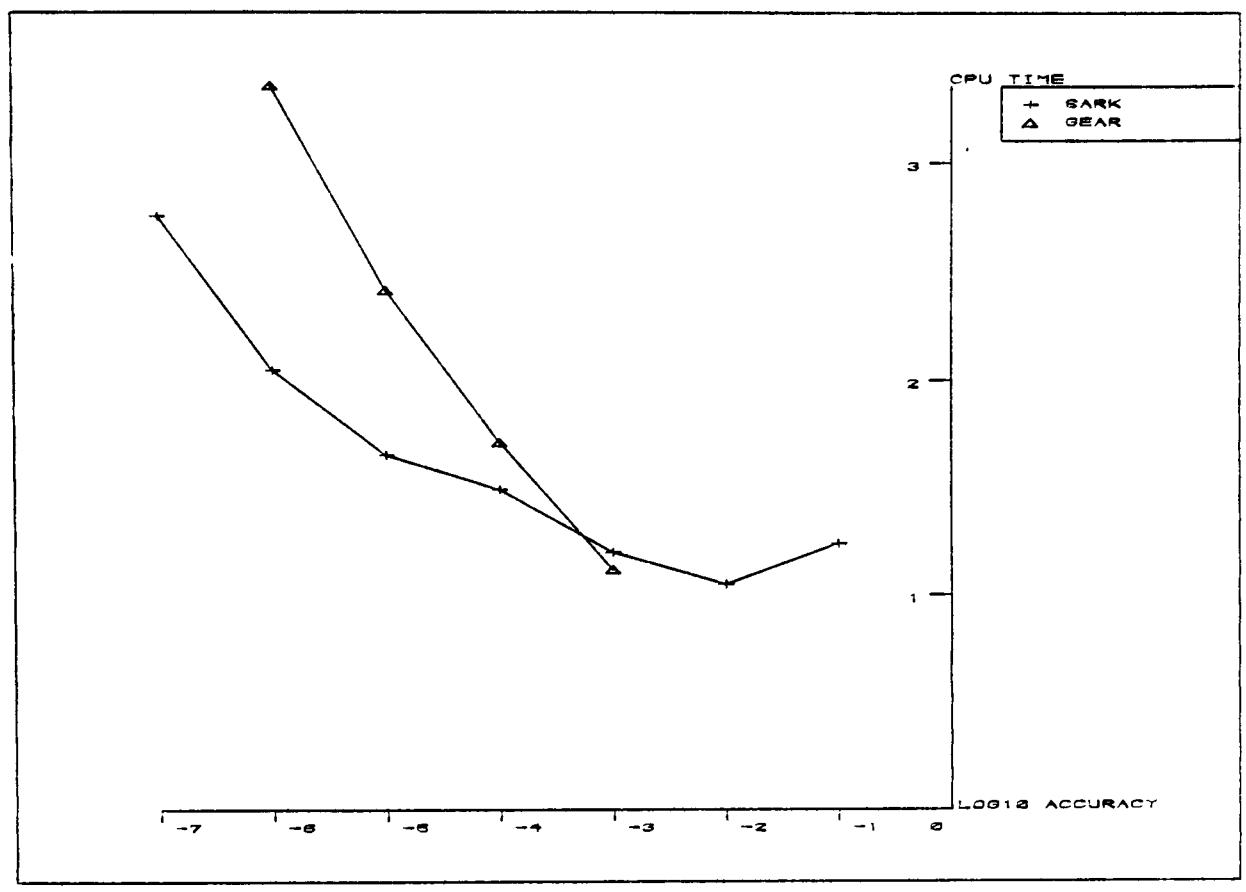


Figure 7.37 : Stiff problem C3

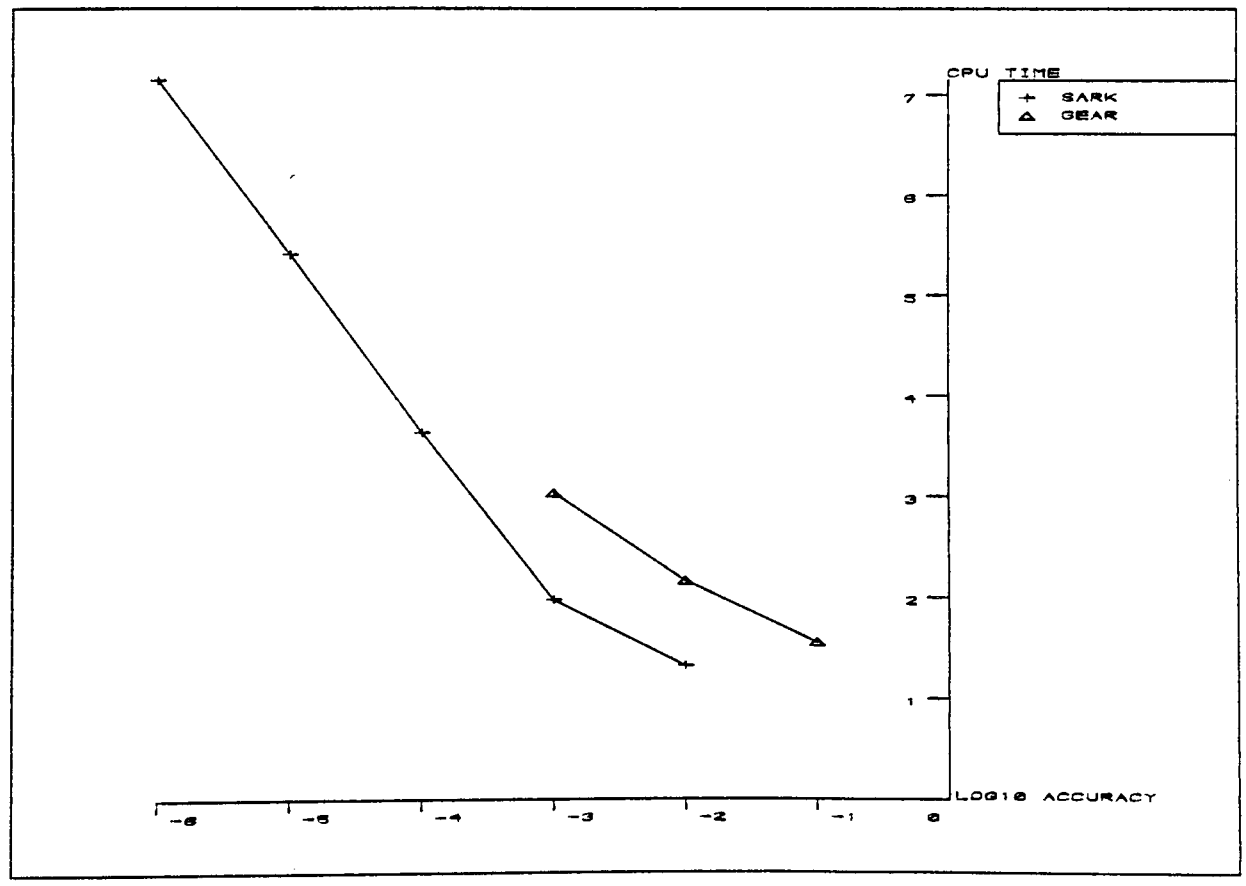


Figure 7.38 : Stiff problem C4

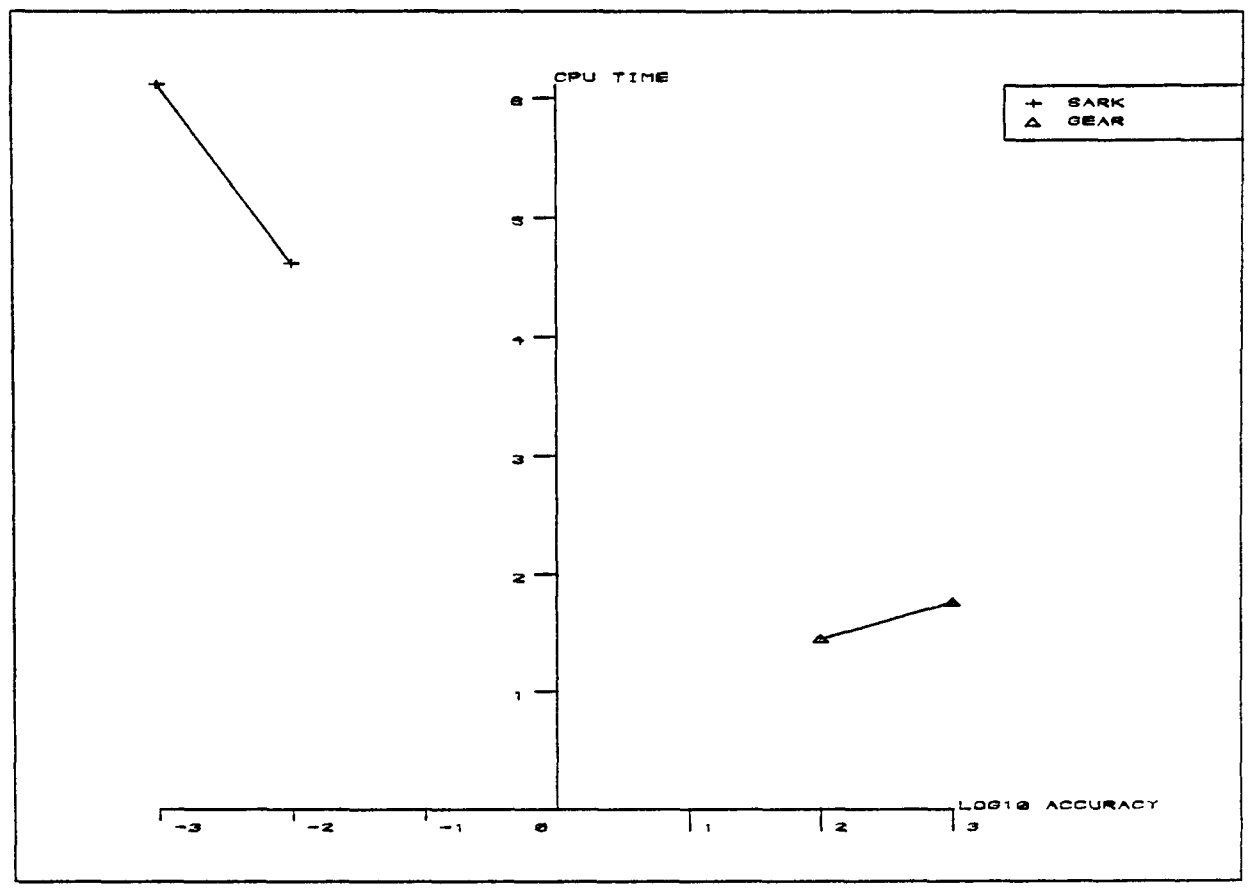


Figure 7.39 : Stiff problem C5

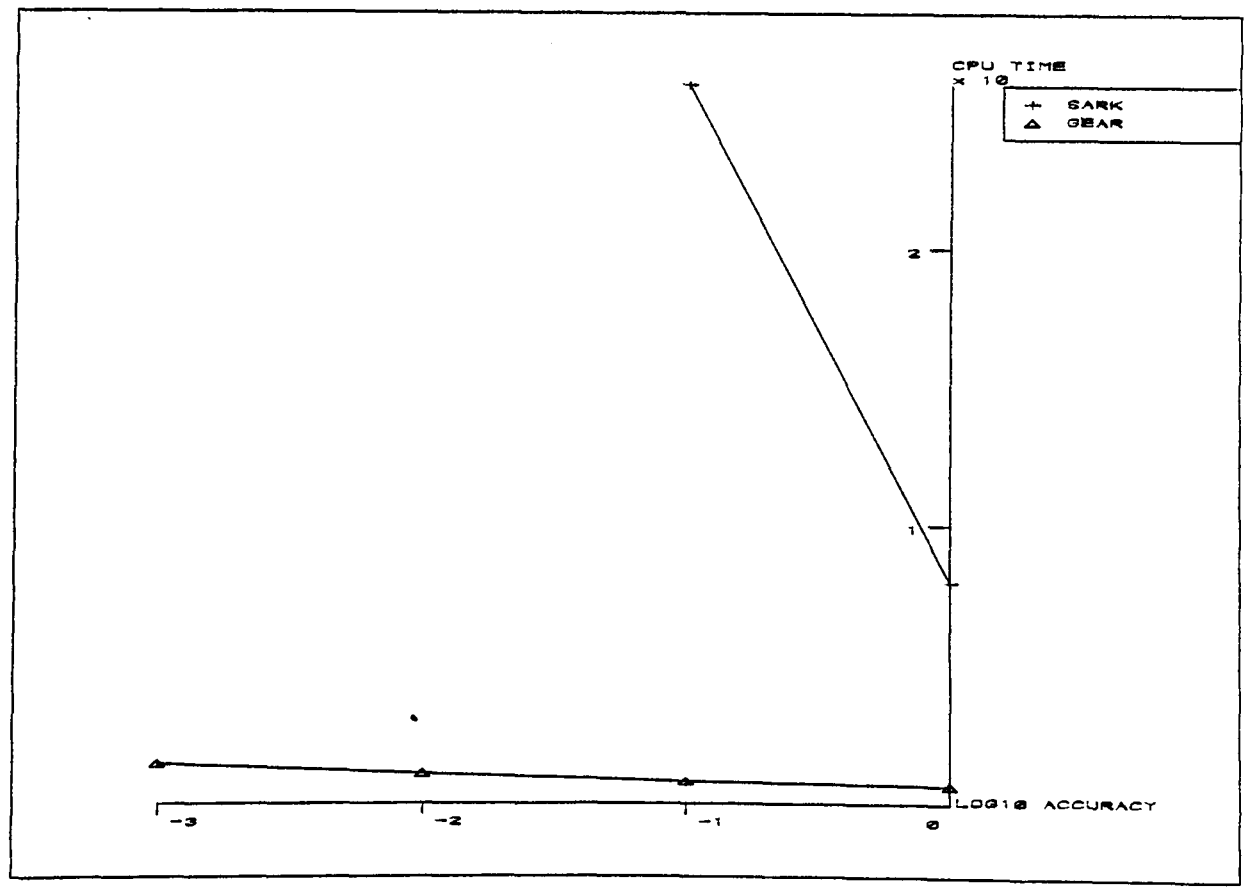


Figure 7.40 : Stiff problem D1

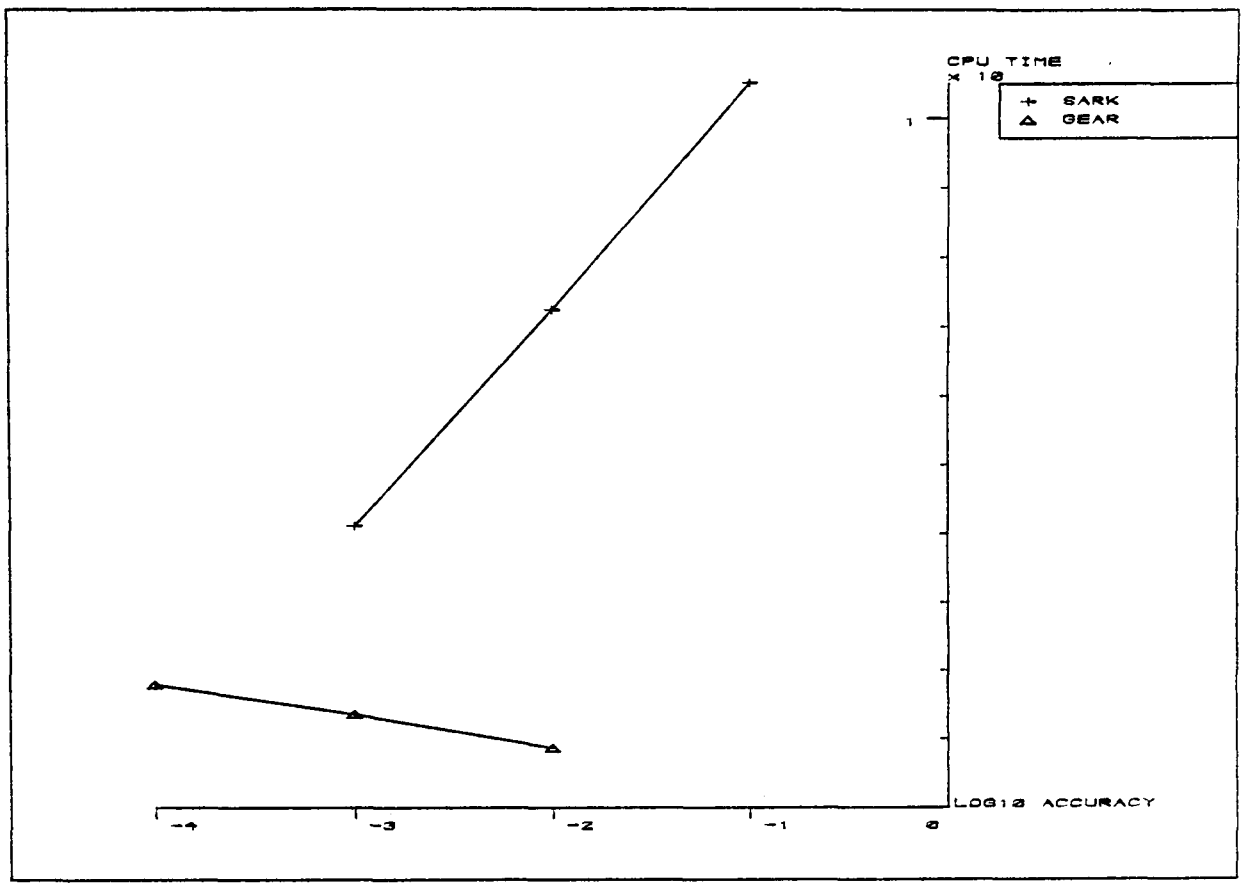


Figure 7.41 : Stiff problem D2

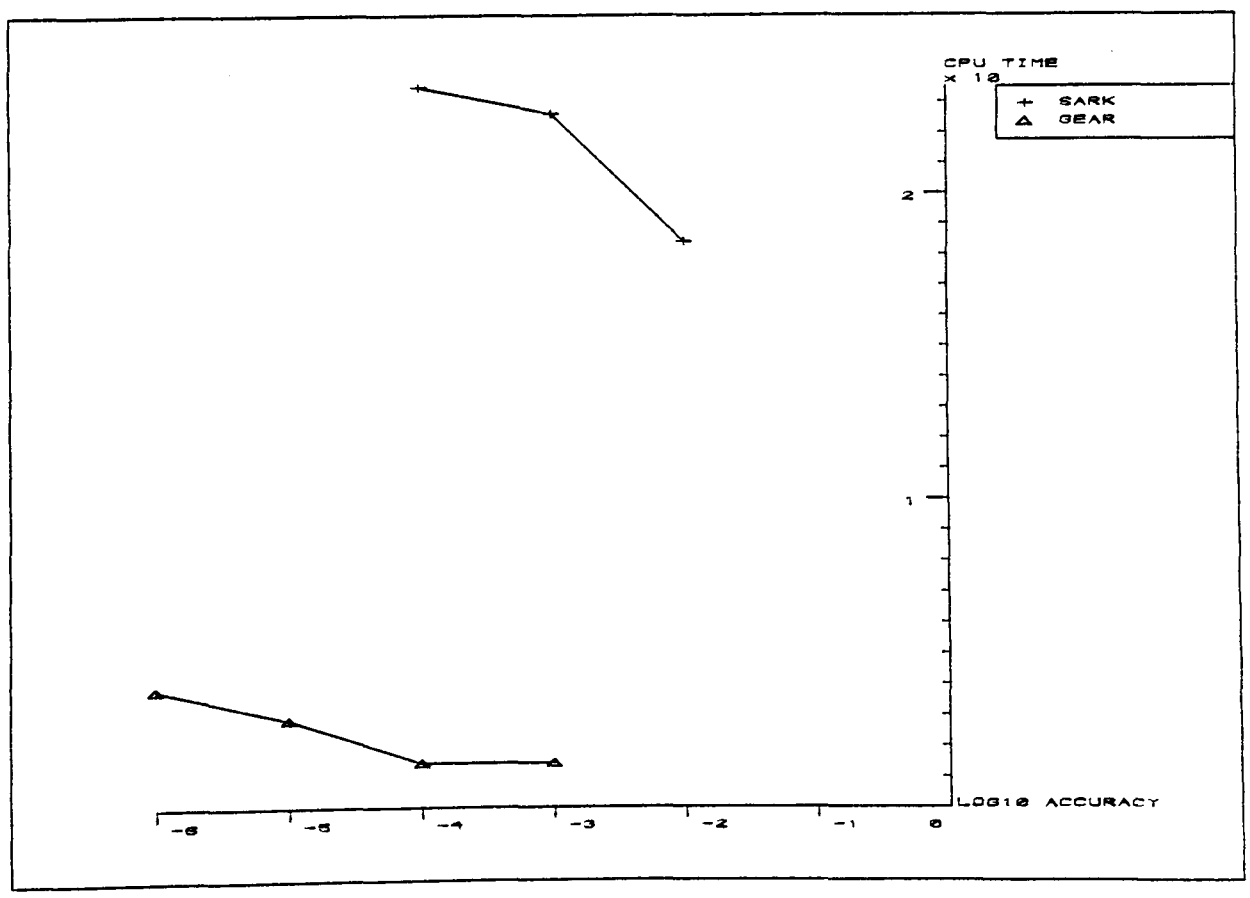


Figure 7.42 : Stiff problem D3

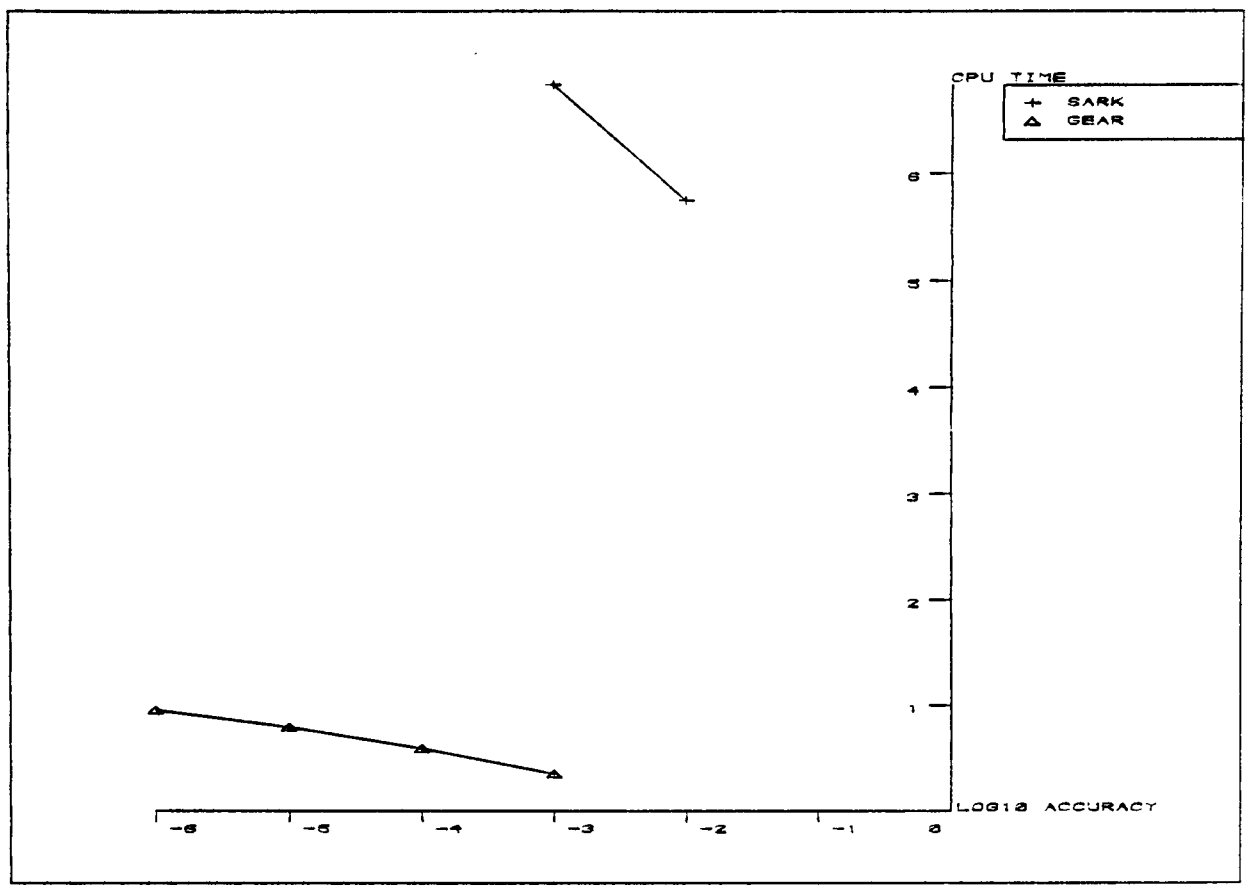


Figure 7.43 : Stiff problem D6

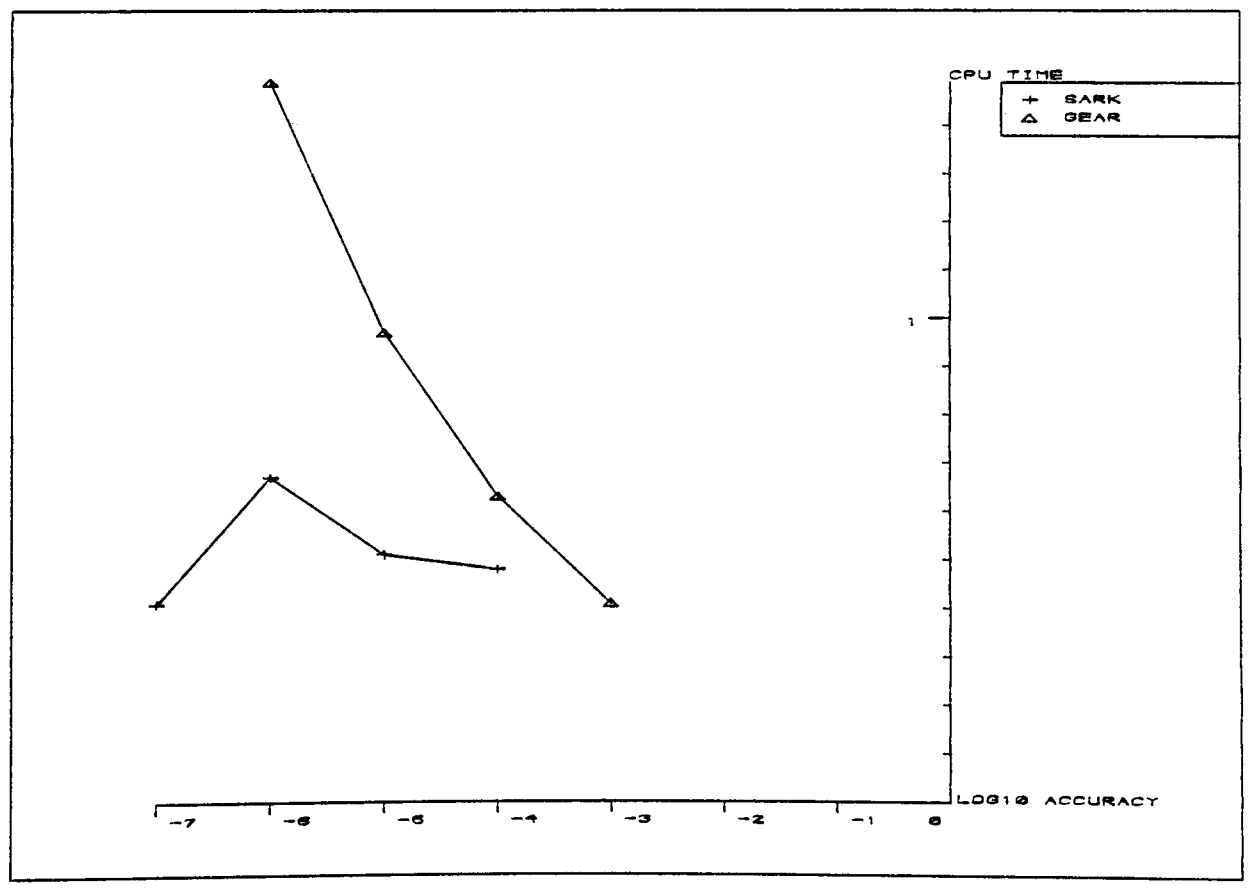


Figure 7.44 : Stiff problem E1

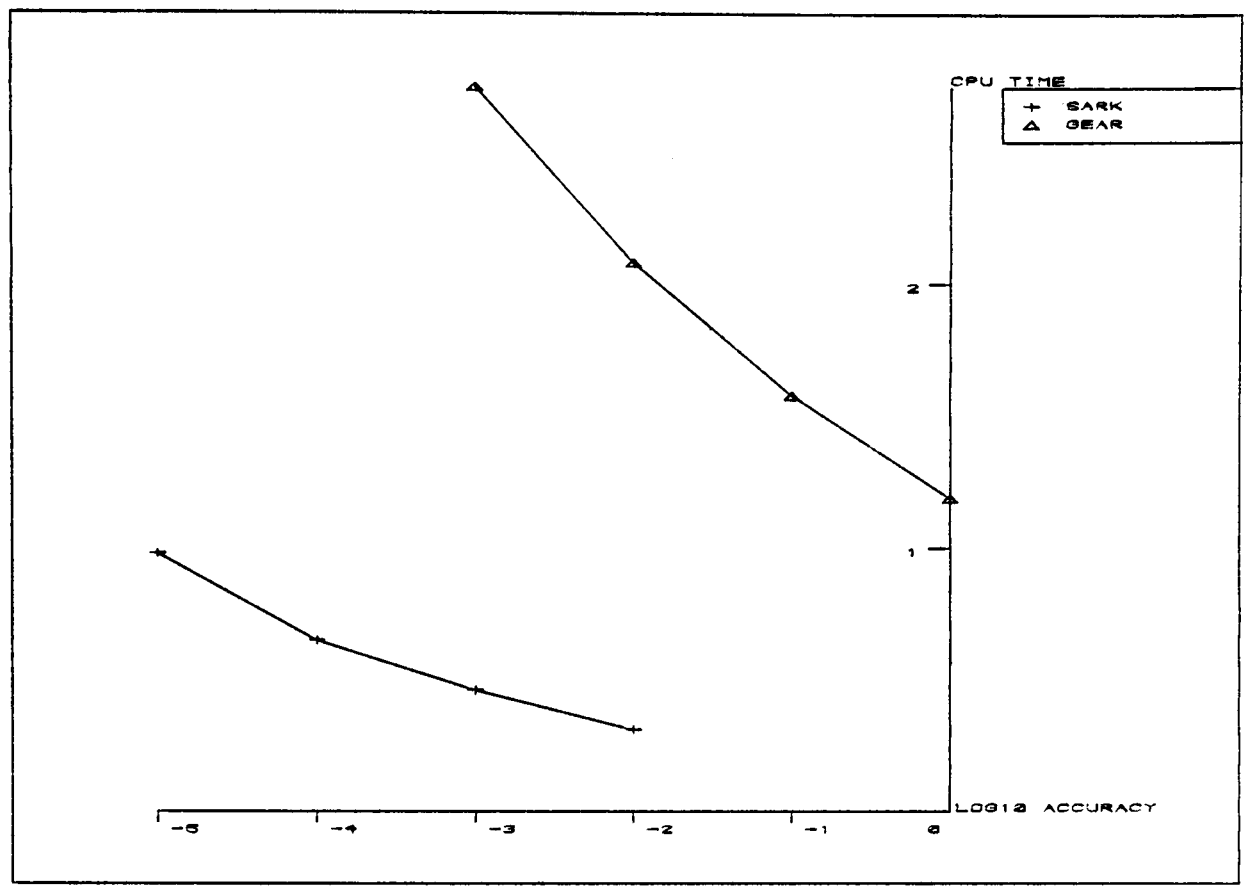


Figure 7.45 : Stiff problem E2

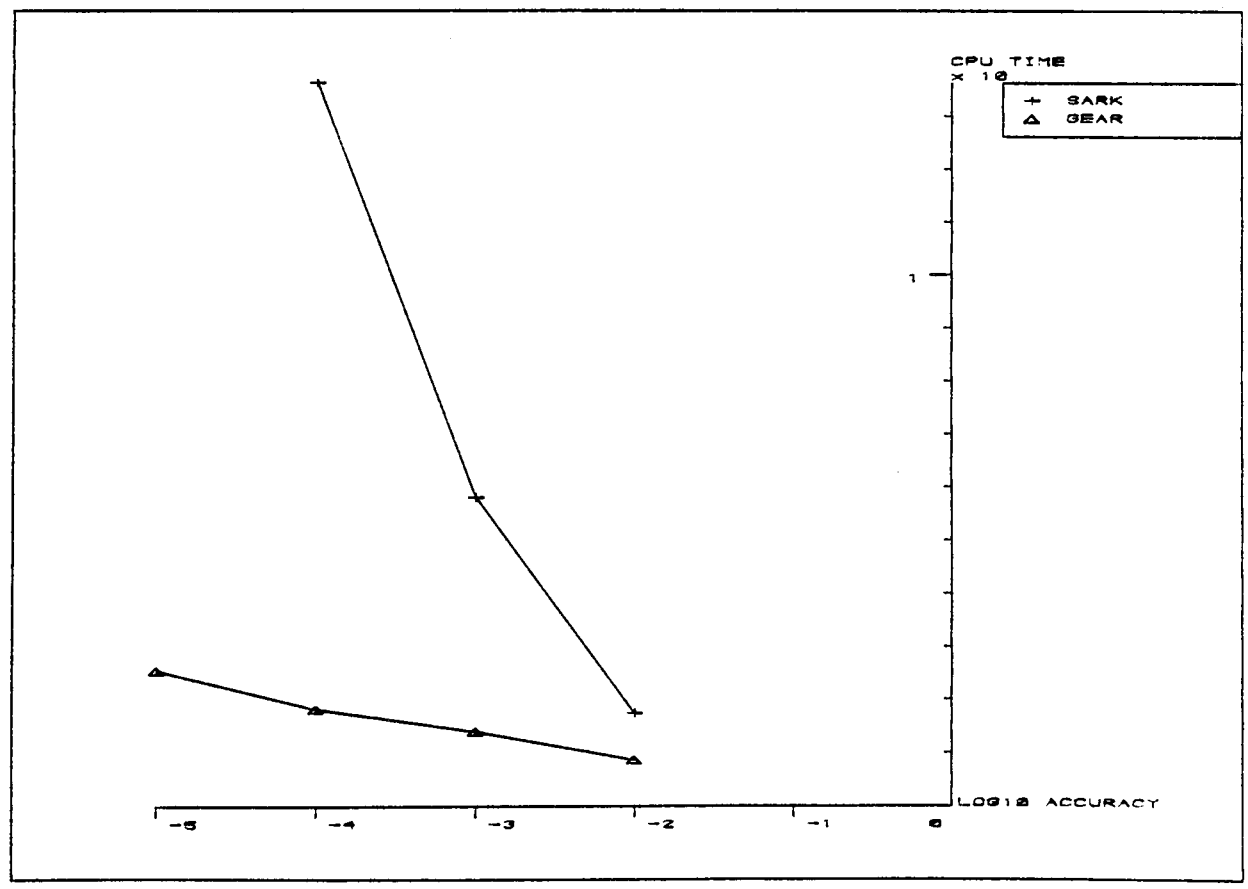


Figure 7.46 : Stiff problem E3

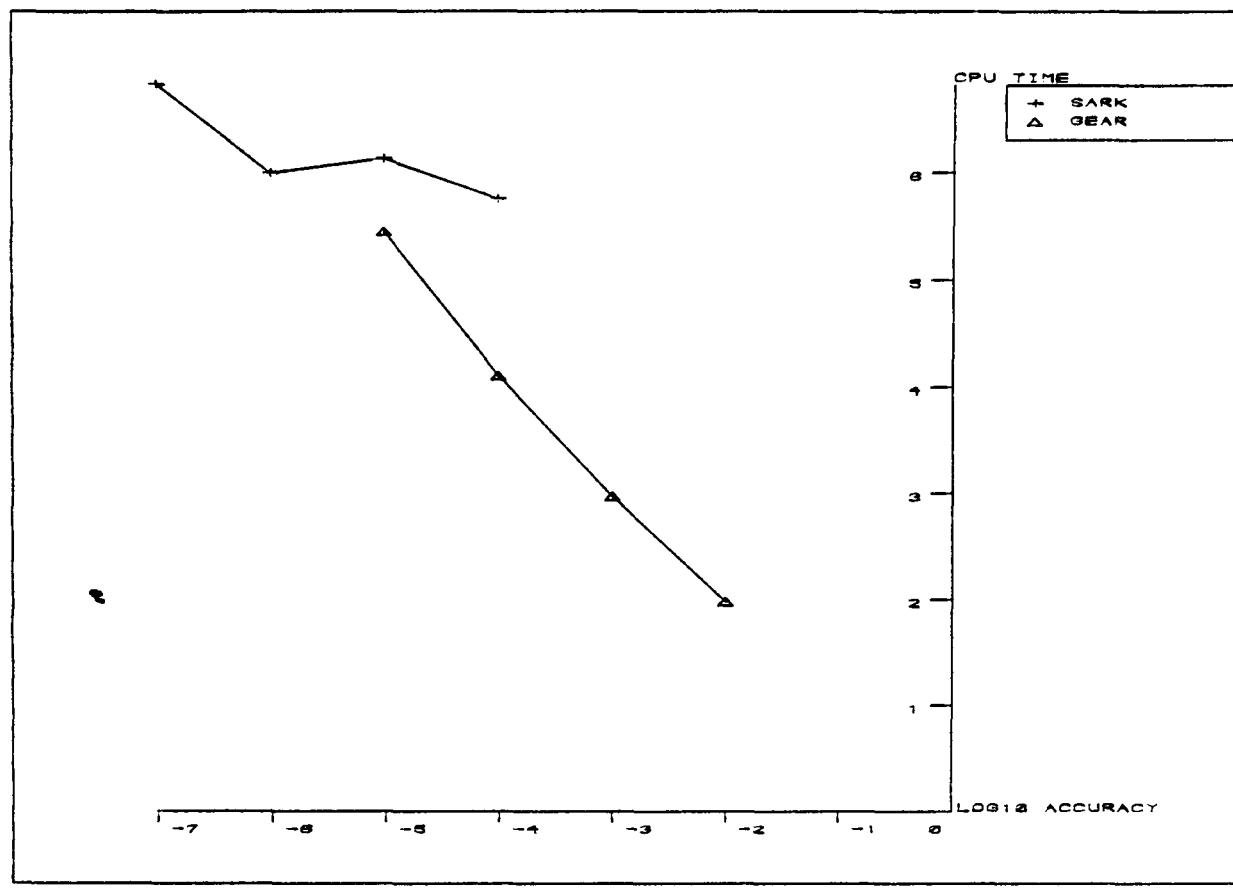


Figure 7.47 : Stiff problem E4

Chapter 8 : CONCLUSIONS

This chapter summarises the results of the previous chapters and makes recommendations for future work.

The stability of a numerical method is an area that has received much attention in recent years with many new stability theories being proposed. The first recognized stability theory concerned A-stability, although this has been severely criticised it is the most practical and hence most widely used. The modulus and argument plots of chapter 2 are a logical extension of this and can be used to evaluate the potential performance of a method in much greater detail. In particular the L-stability of the method can be assessed without detailed analysis being required.

The modulus and argument plots show that the assumption that a precisely A-stable method must be employed for highly oscillatory problems is invalid. To follow correctly the frequency and amplitude of an oscillatory component an explicit method is computationally the cheapest.

The main objective of this work was to develop a type-insensitive code. Initially it was expected that three basic integrators were required to accommodate non-stiff, stiff and oscillatory classifications. For the non-stiff case the obvious choice was an explicit method in this case explicit Runge-Kutta methods. These are very competitive especially if the function evaluations are inexpensive. To complement them a class of methods derived from MIRK methods were investigated, ie. BRK methods. In chapter 2 it was shown that the MRK methods cannot integrate highly oscillatory problems as efficiently as ERK methods and

hence only two categories of integrators were employed in SARK.

The BRK methods are suitable for solving most stiff systems of ODE as they possess very high damping properties. Although SARK is competitive with GEAR and ADAMS it suffers from two slight handicaps. Firstly the error control policy employed is expensive and secondly the methods are prone to computing singular iteration matrices when high order and/or high stiffness ratios are encountered on some linear problems.

Despite these limitations it was felt that they would complement the ERK methods much better than any other class of implicit method. On reflection, however, an improved version of SARK might include BDF methods to cover cases when the singular iteration matrix problem is encountered. This would enable such problems to be integrated without resorting to very low order methods.

Many of the currently used specialized ODE solvers have been under continuous development for several decades. For example Gear's method was first published in 1969. Thus such codes are at a highly developed state. Type-insensitive codes on the other hand are in their infancy and, including SARK, only three such codes exist. They are, however, now at a stage where they are competitive with specialized ODE solvers. Their importance is elevated if the code is used as a black box inside some other process, eg. solving partial differential equations or boundary value problems, where the characteristics of the equations are unknown.

SARK can correctly deduce the characteristics of the equations and

select the category of integrator and the most appropriate order that will lead to the most efficient solution being produced. The overheads in deducing which method, and order, to employ are relatively inexpensive as the decisions are based upon information which already exists.

When high accuracy is required from the solver a small step is usually required even when the problem is stiff. In such cases there is little point in employing an implicit method as this will be expensive. The code developed automatically accommodates for this in its stiffness detection mechanism and thus some stiff problems are solved more efficiently by retaining the non-stiff solver for as long as possible.

It has long been proposed that if there is any ambivalence in the system of ODE being integrated then a stiff solver should be used. The computational danger of doing this is exemplified in chapter 7 where GEAR is over 6 times slower than SARK over the non-stiff problems.

One important feature of SARK is its compactness, as compared to other codes, this makes it particularly suitable for implementation on a small microcomputer.

On the whole SARK is able to solve most systems of ODEs almost as efficiently as any of its constituent methods. If the characteristics of the problem change within the integration range then SARK will be much more efficient. It is also able to integrate most problems of the DETEST package more efficiently than widely used specialized solvers.

One area where integrators are used extensively is in a continuous system simulation languague. One such highly developed languague is ACSL, Mithcell and Gauthier[1986], this allows the user to specify mathematical models as simple statements. The statements are then converted into a FORTRAN program and linked with various routines, one of which is an integrator. The user must select the most appropriate integrator from those supplied. ACSL supplies eight integrators, these being;

Adams-Moulton, variable step, variable order;

Gear's method, variable step, variable order;

Euler's method, fixed step;

Runge-Kutta, fixed step, 2nd order;

Runge-Kutta, fixed step, 4th order (default);

Runge-Kutta-Fehlberg, variable step, 2nd order;

Runge-Kutta-Fehlberg, variable step, 5th order.

Clearly the selection of the correct method can be a daunting task, especially as the user is usually only interested in the solution produced, ie. the integrator is of secondary importance. A valuable alternative would be to provide a type-insensitive code, eg. SARK, as a default integrator, thus allowing the user more freedom to concentrate on the development of the mathematical model.

REFERENCES

ALEXANDER R [1977]

Diagonally implicit Runge-Kutta methods for stiff systems of ODE. SIAM J. Numer. Anal. 14 1006-1021.

ALT R [1972]

Deux théorèms sur la A-stabilité des schémas de Runge-Kutta simplement implicition, Rev. Française d'Automat. Informat. Recherche Opérationelle, 6, sér. R-3, 99-104

BJUREL G, DAHLQUIST G, LINDBERG B, LINDE S, ODEN L [1970]

Survey of stiff ODE. Dept. of Information Processing, The Royal Institute of Technology, Stockholm, Sweden.

BURDEN RL, FAIRES JD, REYNOLDS AC [1978]

Numerical Analysis. Prindle, Weber & Schmidt.

BUTCHER JC [1963]

Coefficients for the study of Runge-Kutta processess. J. Austral. Math. Soc. 3 185-201.

BUTCHER JC [1964]

On Runge-Kutta processes of high order. J. Austral. Math. Soc. 4 179-193.

BUTCHER JC [1965]

On the attainable order of Runge-Kutta methods. Math. Comput. 19 408-417.

BUTCHER JC [1976]

On the implementation of implicit Runge-Kutta methods. BIT 16 237-240.

BUTCHER JC [1987]

The numerical analysis of ODE, Runge-Kutta and general linear methods. Wiley.

CASH JR [1975]

A class of implicit Runge-Kutta method for the numerical integration of stiff ODE. J. ACM 22 504-511.

CASH JR [1980]

On the integration of stiff system of ODE using extended Backward Differentiation formulae. 34 235-246.

CASH JR [1983]

The integration of stiff initial value problems in ODE using modified extended Backward Differentiation formulae. Comp & Maths. with Appls. 9 645-657.

CHIPMAN FH [1963]

A-stable Runge-Kutta processes. BIT 11 384-388.

CRAIGIE JAI [1975]

A variable order multistep method for the numerical solution of stiff systems of ODE. Numerical Analysis report 11, University of Manchester, Dept. of Mathematics, Manchester.

CROUZEIX M [1976]

Sur les méthodes de Runge-Kutta pour l'approximation des problèmes d'évolution, Lecture notes in Econom. and Maths. Systems No. 134, Springer Berlin 206-223.

DAHLQUIST G [1963]

A special stability problem for linear multistep methods. BIT 3 27-43.

DEKKER K, VERWER JG [1984]

Stability of Runge-Kutta methods for stiff non-linear differential equations, North-Holland, Amsterdam.

EHLE BL [1968]

High order A-stable methods for the numerical integration of differential equations. BIT 18 276-278.

ENRIGHT WH, PRYCE JD [1983]

Two Fortran packages for assessing initial value problems. Technical report #167/83, Dept. of Computer Science, University of Toronto, Toronto, Canada.

FEHLBERG E [1968]

Klassiche Runge-Kutta-Formeln vierter und niedrigere probleme, Computimg, 6 61-71.

GEAR WC [1971]

Numerical initial value problems in ODE. Prentice-Hall.

GEAR WC [1981]

Numerical solution of ODE : Is there anything left to do? SIAM Rev. 23 19-24.

GLADWELL IG [1974]

Initial value routines in the NAG library. ACM Trans. Maths. Soft. 5 386-400.

HERICI P [1962]

Discrete variable methods in ODE. Wiley.

HINDMARSH AC [1974]

Gear : ODE system solver. UCID 30001 Rev. 2. Lawerence Livermore, Univ. of California.

JELTSH AA [1978]

Stability on the imaginary axis and A-stability of linear multistep methods. BIT 18 170-174.

LAMBERT JD [1973]

Computational methods in ODE. Wiley.

LAMBERT JD [1979]

Stiffness. NA report 37, University of Dundee.

LAWSON JD [1967]

An order 5 Runge-Kutta process with extended region of stability. SIAM J. Numer. Anal. 3 593-345.

MITCHELL EEL, GAUTHIER JS [1986]

ACSL Reference Manuael, Mitchell and Gauthier Associates, Concord, Mass., USA.

NORSETT SP [1974]

One-step methods of Hermite type for the numerical integration of stiff systems, BIT 14 63-77.

NORSETT SP, THOMSEN PG [1986]

Switching between modified and fix-point iteration for implicit ODE solvers. BIT 26 339-345.

PETZOLD L [1983]

Automatic selection of methods for solving stiff and non-stiff systems of ODE. SIAM J. Sci. Stat. Comput. 4 136-148.

PROTHERO A, ROBINSON A [1974]

On the stability and accuracy of one-step methods for solving stiff systems of ODE. Math. Comput. 25 145-162.

RICHARDS CW, EVERETT MG [1984]

Backward Runge-Kutta methods for stiff systems of ODE. Submitted to IMA J. Numer. Anal.

SCHIED RE [1983]

The accurate numerical solution of highly oscillatory ODE. Math. Comput. 41 487-509.

SHAMPINE LF [1975]

Stiffness and non-stiff differential equation solvers. Numerische Behandglung vaon Differertialgleichungen. ed. L COLLATZ, Inst. series Numer. Math. 27 287-301 Birkhauser Basel Switz.

SHAMPINE LF, GORDON MK [1975]

Computer solution of ODE. WH Freeman.

SHAMPINE LF, HIEBERT KL [1977]

Detecting stiffness with the Fehlberg (4,5) formulae. Comp. & Math. with Appls. 3 41-46.

SINGHAL A [1980]

Implicit Runge-Kutta formulae for the intregration of ODE. PhD. Thesis, University of London.

SKELL RD, KONG AK [1977]

Blended Linear Multistep methods. ACM Trans. on Maths. Soft. 3 326-345.

VERNER SH [1978]

Explicit Runge-Kutta methods with estimates of the local truncation error. SIAM J. Numer. Anal. 15 772-790.

WANNER G, HAIRER E, NORSETT SP [1978]

Order stars and stability theorems. BIT 18 475-489.

Appendix A : Normalized results for the non-stiff problems of DETEST

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-4	63	0	10	0.06
	-5	96	0	15	0.08
	-6	143	0	22	0.12
	-7	212	0	34	0.17
	-8	320	0	52	0.22
GEAR	-3	109	8	34	0.20
	-4	109	11	53	0.31
	-5	130	12	72	0.40
	-6	160	14	97	0.52
	-7	207	18	137	0.71
ADAMS	-3	89	-	39	0.18
	-4	105	-	52	0.20
	-5	124	-	68	0.28
	-6	167	-	100	0.39
	-7	212	-	134	0.49

Table A.1 : Problem A1 (Figure 7.1)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-3	48	0	6	0.03
	-4	54	0	8	0.04
	-5	74	0	11	0.05
	-6	102	0	16	0.08
	-7	138	0	22	0.10
	-8	189	0	30	0.13
GEAR	-4	99	12	47	0.28
	-5	131	16	71	0.40
	-6	161	15	101	0.53
	-7	213	16	146	0.77
ADAMS	-3	59	-	28	0.12
	-4	78	-	39	0.16
	-5	99	-	52	0.21
	-6	121	-	69	0.28
	-7	146	-	94	0.36

Table A.2 : Problem A2 (Figure 7.2)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-2	207	0	26	0.16
	-3	324	0	45	0.25
	-4	539	0	75	0.46
	-5	883	0	127	0.70
	-6	1377	0	214	1.08
GEAR	-2	262	23	129	0.78
	-3	321	25	196	1.09
	-4	442	35	303	1.60
	-5	599	38	453	2.30
ADAMS	-2	188	-	99	0.40
	-3	235	-	137	0.54
	-4	329	-	202	0.79
	-5	415	-	278	1.10
	-6	453	-	328	1.30

Table A.3 : Problem A3 (Figure 7.3)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-3	41	0	4	0.03
	-4	60	0	8	0.04
	-5	85	0	13	0.06
	-6	145	0	21	0.11
	-7	214	0	33	0.15
	-8	336	0	54	0.24
GEAR	-2	63	8	24	0.16
	-3	97	10	37	0.25
	-4	118	10	52	0.33
	-5	139	11	70	0.41
	-6	176	15	101	0.55
ADAMS	-2	60	-	19	0.10
	-3	81	-	30	0.15
	-4	97	-	42	0.21
	-5	127	-	61	0.27
	-6	131	-	70	0.29

Table A.4 : Problem A4 (Figure 7.4)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-3	43	0	6	0.03
	-4	65	0	9	0.05
	-5	105	0	16	0.08
	-6	166	0	26	0.12
	-7	264	0	43	0.19
GEAR	-2	83	10	33	0.22
	-3	110	13	49	0.30
	-4	149	14	72	0.42
	-5	174	14	96	0.54
ADAMS	-2	54	-	22	0.11
	-3	72	-	32	0.16
	-4	93	-	45	0.19
	-5	106	-	57	0.23
	-6	121	-	71	0.29

Table A.5 : Problem A5 (Figure 7.5)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-2	437	0	60	0.39
	-3	701	0	101	0.61
	-4	1058	0	165	1.07
	-5	1699	0	277	1.77
GEAR	-1	443	29	229	1.82
	-2	553	37	352	2.56
	-3	738	44	538	3.73
	-4	1077	57	855	5.73
ADAMS	-1	316	-	180	0.85
	-2	404	-	254	1.17
	-3	461	-	330	1.52
	-4	608	-	463	2.17
	-5	705	-	579	2.79

Table A.6 : Problem B1 (Figure 7.6)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-4	145	0	22	0.18
	-5	174	0	26	0.21
	-6	219	0	34	0.26
	-7	291	0	47	0.36
	-8	427	0	70	0.54
GEAR	-3	111	11	44	0.45
	-4	155	15	67	0.68
	-5	190	16	94	0.92
	-6	240	20	129	1.43
	-7	300	21	186	1.72
ADAMS	-3	183	-	106	0.50
	-4	196	-	112	0.56
	-5	210	-	121	0.76
	-6	220	-	141	0.81
	-7	340	-	253	1.32

Table A.7 : Problem B2 (Figure 7.7)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-4	90	0	14	0.11
	-5	125	0	19	0.16
	-6	175	0	28	0.22
	-7	253	0	41	0.32
	-8	363	0	59	0.45
	-9	520	0	85	0.66
GEAR	-3	113	11	43	0.45
	-4	141	11	64	0.63
	-5	179	14	88	0.86
	-6	229	16	119	1.17
	-7	279	18	168	1.56
ADAMS	-3	94	-	46	0.26
	-4	113	-	59	0.34
	-5	126	-	70	0.41
	-6	159	-	98	0.58
	-7	229	-	150	0.86

Table A.8 : Problem B3 (Figure 7.8)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-1	209	0	32	0.27
	-2	313	0	49	0.38
	-3	459	0	73	0.55
	-4	671	0	108	0.82
	-5	965	0	157	1.18
	-6	1369	0	227	1.66
	-7	1987	0	330	2.48
GEAR	-1	222	12	130	1.17
	-2	290	15	176	1.58
	-3	355	18	230	2.08
	-4	469	24	333	2.95
	-5	649	32	491	4.24
ADAMS	-1	160	-	78	0.47
	-2	192	-	113	0.68
	-3	249	-	170	1.00
	-4	325	-	238	1.43
	-5	406	-	320	1.96

Table A.9 : Problem B4 (Figure 7.9)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-3	296	0	38	0.31
	-4	409	0	62	0.44
	-5	647	0	106	0.71
	-6	1099	0	182	1.25
GEAR	-2	247	15	126	1.21
	-3	292	15	170	1.60
	-4	380	19	242	2.20
	-5	506	25	352	3.10
	-6	707	34	525	4.50
ADAMS	-2	171	-	87	0.53
	-3	199	-	122	0.70
	-4	253	-	169	0.97
	-5	316	-	227	1.53
	-6	377	-	288	1.99

Table A.10 : Problem B5 (Figure 7.10)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-4	106	0	16	0.25
	-5	157	0	25	0.38
	-6	241	0	39	0.59
	-7	375	0	61	0.92
	-8	593	0	97	1.45
GEAR	-3	200	11	48	1.57
	-4	226	11	69	2.08
	-5	294	14	95	2.85
	-6	371	17	138	3.86
	-7	438	19	198	5.19
ADAMS	-3	121	-	58	0.70
	-4	139	-	64	0.82
	-5	155	-	87	1.12
	-6	201	-	122	1.61
	-7	314	-	225	2.98

Table A.11 : Problem C1 (Figure 7.11)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-4	315	0	50	0.82
	-5	324	0	52	0.84
	-6	375	0	59	0.96
	-7	480	0	77	1.24
	-8	663	0	107	1.72
GEAR	-3	203	11	49	1.65
	-4	262	14	74	2.46
	-5	306	15	104	3.15
	-6	386	18	147	4.21
	-7	473	21	207	5.61
ADAMS	-3	433	-	280	2.56
	-4	463	-	294	2.81
	-5	489	-	309	3.07
	-6	541	-	344	3.60
	-7	557	-	365	4.10

Table A.12 : Problem C2 (Figure 7.12)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-4	163	0	24	0.40
	-5	182	0	27	0.45
	-6	215	0	33	0.53
	-7	273	0	44	0.71
	-8	410	0	67	1.16
GEAR	-3	169	10	39	1.31
	-4	228	12	60	1.95
	-5	303	16	88	2.76
	-6	372	19	122	3.67
	-7	436	21	172	4.87
ADAMS	-3	219	-	129	1.28
	-4	235	-	142	1.49
	-5	248	-	144	1.64
	-6	307	-	186	2.31
	-7	341	-	214	2.69

Table A.13 : Problem C3 (Figure 7.13)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-4	163	0	24	1.73
	-5	182	0	27	1.95
	-6	214	0	32	2.31
	-7	271	0	44	2.99
	-8	408	0	67	4.53
GEAR	-3	681	12	38	23.57
	-4	792	13	61	29.91
	-5	896	14	87	36.52
	-6	1141	19	122	47.34
ADAMS	-2	216	-	131	4.66
	-3	210	-	129	4.90
	-4	223	-	135	5.77
	-5	231	-	148	6.65
	-6	262	-	156	8.37
	-7	352	-	225	13.26

Table A.14 : Problem C4 (Figure 7.14)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-3	49	0	7	0.61
	-4	75	0	11	0.98
	-5	112	0	17	1.42
	-6	172	0	27	2.14
	-7	270	0	44	3.37
GEAR	-1	217	6	15	6.12
	-2	228	6	21	7.07
	-3	299	7	30	9.43
	-4	390	10	41	12.04
	-5	430	11	53	13.89
ADAMS	-1	35	-	11	0.68
	-2	54	-	17	1.05
	-3	66	-	25	1.44
	-4	80	-	37	1.91

Table A.15 : Problem C5 (Figure 7.15)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-1	238	0	38	0.33
	-2	358	0	58	0.49
	-3	546	0	90	0.75
	-4	837	0	138	1.17
	-5	1283	0	212	1.79
GEAR	-1	288	14	121	1.57
	-2	322	18	167	2.23
	-3	412	21	246	2.87
	-4	559	27	371	4.22
ADAMS	-1	152	-	72	0.51
	-2	203	-	112	0.77
	-3	261	-	172	1.28
	-4	363	-	276	1.94
	-5	416	-	327	2.27

Table A.16 : Problem D1 (Figure 7.16)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-1	297	0	42	0.39
	-2	378	0	58	0.52
	-3	492	0	81	0.70
	-4	697	0	115	0.96
	-5	1014	0	168	1.39
	-6	1468	0	243	2.01
GEAR	-1	334	22	144	1.92
	-2	408	26	208	2.91
	-3	522	26	321	3.80
	-4	743	37	500	5.78
ADAMS	0	189	-	89	0.64
	-1	224	-	130	0.87
	-2	277	-	178	1.16
	-3	338	-	243	1.62
	-4	407	-	312	2.15
	-5	510	-	412	2.95

Table A.17 : Problem D2 (Figure 7.17)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-1	325	0	41	0.42
	-2	489	0	68	0.64
	-3	719	0	115	0.97
	-4	1212	0	201	1.91
	-5	2105	0	349	2.89
GEAR	0	397	28	173	2.44
	-1	493	34	252	2.35
	-2	646	39	382	4.95
	-3	860	47	568	6.48
	-4	1223	58	882	9.76
ADAMS	0	258	-	142	1.10
	-1	325	-	209	1.50
	-2	417	-	298	2.10
	-3	556	-	444	3.22
	-4	665	-	556	4.07

Table A.18 : Problem D3 (Figure 7.18)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	0	429	0	53	0.63
	-1	645	0	82	0.83
	-2	883	0	129	1.18
	-3	1270	0	210	1.82
	-4	2136	0	355	3.07
GEAR	0	563	45	264	3.33
	-1	742	55	400	5.01
	-2	1037	61	671	7.88
	-3	1524	75	1103	12.51
ADAMS	0	352	-	208	1.48
	-1	459	-	325	2.31
	-2	592	-	461	3.46
	-3	786	-	656	4.98

Table A.19 : Problem D4 (Figure 7.19)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	0	909	0	113	1.18
	-1	1244	0	163	1.76
	-2	1506	0	233	2.05
	-3	2147	0	354	2.93
	-4	3291	0	547	4.51
GEAR	0	1030	84	551	8.17
	-1	1500	93	981	13.28
ADAMS	0	595	-	432	2.99
	-1	851	-	697	5.18

Table A.20 : Problem D5 (Figure 7.20)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-3	156	0	25	0.15
	-4	245	0	39	0.23
	-5	384	0	63	0.38
	-6	611	0	100	0.59
	-7	974	0	161	0.89
GEAR	-2	133	8	64	0.59
	-3	174	11	95	0.85
	-4	216	14	132	1.14
	-5	289	17	202	1.66
	-6	404	23	306	2.42
ADAMS	-2	108	-	46	0.24
	-3	137	-	66	0.33
	-4	169	-	90	0.45
	-5	201	-	119	0.68
	-6	250	-	172	1.09

Table A.21 : Problem E1 (Figure 7.21)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-2	457	0	57	0.39
	-3	614	0	82	0.53
	-4	860	0	120	0.74
	-5	1158	0	178	1.01
	-6	1670	0	269	1.47
	-7	2478	0	407	2.20
GEAR	-2	508	36	271	2.43
	-3	578	37	360	3.17
	-4	710	42	504	4.26
	-5	966	54	744	6.02
	-6	1353	71	1097	8.62
ADAMS	-2	344	-	215	0.95
	-3	410	-	292	1.29
	-4	525	-	410	1.85
	-5	666	-	543	2.49
	-6	855	-	726	3.39

Table A.22 : Problem E2 (Figure 7.22)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-2	333	0	47	0.35
	-3	506	0	71	0.53
	-4	740	0	108	0.78
	-5	1046	0	161	1.11
	-6	1452	0	240	1.57
	-7	2169	0	360	2.44
GEAR	-2	267	13	164	1.45
	-3	349	17	233	2.08
	-4	487	24	359	3.03
	-5	710	35	559	4.56
	-6	1016	50	841	6.72
ADAMS	-2	173	-	99	0.51
	-3	227	-	151	0.74
	-4	322	-	235	1.20
	-5	390	-	301	1.61
	-6	501	-	404	2.29

Table A.23 : Problem E3 (Figure 7.23)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-4	39	0	4	0.05
	-5	57	0	7	0.06
	-6	88	0	12	0.08
	-7	133	0	20	0.12
	-8	210	0	33	0.19
GEAR	-1	37	5	9	0.13
	-2	52	5	16	0.20
	-3	65	6	23	0.25
	-4	85	8	34	0.36
	-5	112	9	47	0.49
ADAMS	-1	28	-	7	0.05
	-2	35	-	11	0.07
	-3	48	-	17	0.10
	-4	63	-	24	0.14
	-5	78	-	33	0.19

Table A.24 : Problem E4 (Figure 7.24)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-3	45	0	3	0.04
	-4	57	0	6	0.05
	-5	66	0	10	0.07
	-6	109	0	17	0.12
	-7	177	0	28	0.19
GEAR	-1	68	6	18	0.25
	-2	107	9	33	0.41
	-3	141	12	46	0.55
	-4	170	13	64	0.71
	-5	207	14	92	0.92
ADAMS	-1	56	-	12	0.10
	-2	74	-	21	0.15
	-3	89	-	29	0.19
	-4	106	-	41	0.25
	-5	137	-	61	0.35

Table A.25 : Problem E5 (Figure 7.25)

Appendix B : Normalized results for the stiff problems of DETEST

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-4	557	6	44	0.57
	-5	745	4	68	0.73
	-6	1143	4	135	1.28
	-7	1753	4	243	2.04
	-8	2520	4	378	2.91
GEAR	-3	166	15	68	1.14
	-4	229	19	103	1.66
	-5	272	21	137	2.16
	-6	379	26	203	2.97

Table B.1 : Problem A1 (Figure 7.26)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-1	5458	85	244	2.30
	-2	5532	81	243	3.34
	-3	3637	52	199	5.45
GEAR	-3	324	21	85	3.88
	-4	416	25	134	5.59
	-5	528	29	190	7.70
	-6	658	33	261	10.14
	-7	827	40	362	12.96

Table B.2 : Problem A2 (Figure 7.27)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-3	723	9	59	0.81
	-4	1178	15	15	1.17
GEAR	-3	303	28	129	2.23
	-4	317	23	164	2.33

Table B.3 : Problem A3 (Figure 7.28)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	0	1286	10	60	3.53
	-1	1355	10	68	3.75
	-2	1433	10	80	3.99
	-3	1512	10	92	4.23
GEAR	-3	413	26	113	6.00
	-4	511	30	165	8.07

Table B.4 : Problem A4 (Figure 7.29)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	0	4097	16	401	4.30
	-1	5398	16	572	5.43
	-2	5858	15	660	6.08
	-3	5862	13	710	6.46
	-4	5870	8	812	6.74
GEAR	-2	672	41	367	5.19
	-3	879	47	564	7.38
	-4	1186	62	833	10.42

Table B.5 : Problem B1 (Figure 7.30)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-4	402	0	65	0.62
	-5	433	0	70	0.64
	-6	490	0	79	0.76
	-7	582	0	94	0.94
	-8	730	0	119	1.12
GEAR	-3	182	14	62	1.60
	-4	233	16	91	2.25
	-5	287	19	125	2.89
	-6	352	20	172	3.76
	-7	438	23	249	5.22

Table B.6 : Problem B2 (Figure 7.31)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-4	504	0	81	0.80
	-5	548	0	88	0.84
	-6	619	0	100	0.95
	-7	734	0	119	1.17
	-8	928	0	152	1.46
GEAR	-3	191	14	68	1.78
	-4	249	17	99	2.37
	-5	318	20	143	3.20
	-6	383	22	192	4.20
	-7	465	24	266	5.32

Table B.7 : Problem B3 (Figure 7.32)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-3	620	4	55	0.76
	-4	754	4	76	1.01
	-5	1021	2	136	1.48
	-6	1429	0	237	2.26
	-7	1637	0	271	2.54
	-8	1959	0	325	3.02
GEAR	-3	261	16	114	2.52
	-4	338	21	156	3.45
	-5	445	24	230	4.76
	-6	606	31	347	6.85

Table B.8 : Problem B4 (Figure 7.33)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-1	2176	6	282	3.10
	-2	4511	5	659	6.76
	-3	2584	4	357	4.05
	-4	1934	5	243	3.02
	-5	2826	9	356	4.01
	-6	3718	6	542	5.50
GEAR	-2	3304	129	2343	40.87
	-3	3451	138	2331	43.92
	-4	3558	141	2466	45.71
	-5	3817	151	2663	49.34

Table B.9 : Problem B5 (Figure 7.34)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-3	491	9	45	0.65
	-4	667	8	65	1.01
	-5	752	5	101	0.94
	-6	1209	10	146	1.41
	-7	948	4	133	1.14
	-8	916	0	151	1.10
GEAR	-3	190	18	81	1.31
	-4	253	21	123	1.90
	-5	324	26	172	2.49
	-6	400	28	239	3.33

Table B.10 : Problem C1 (Figure 7.35)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-4	1148	18	46	1.00
	-5	999	11	65	0.99
	-6	1050	5	116	1.23
	-7	1637	6	214	1.94
GEAR	-3	183	17	76	1.20
	-4	253	21	117	1.79
	-5	324	25	166	2.46
	-6	409	25	238	3.34

Table B.11 : Problem C2 (Figure 7.36)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-1	1504	19	51	1.24
	-2	1501	16	60	1.05
	-3	1518	16	79	1.20
	-4	1540	16	105	1.49
	-5	1522	9	158	1.64
	-6	1724	5	226	2.04
	-7	2216	4	315	2.75
GEAR	-3	169	16	71	1.12
	-4	240	20	109	1.71
	-5	322	25	161	2.41
	-6	441	27	243	3.35

Table B.12 : Problem C3 (Figure 7.37)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-2	1383	9	78	1.32
	-3	1799	8	156	1.97
	-4	3040	6	419	3.63
	-5	4448	5	690	5.41
	-6	5890	5	930	7.15
GEAR	-1	232	20	101	1.55
	-2	306	20	146	2.16
	-3	415	27	204	3.03

Table B.13 : Problem C4 (Figure 7.38)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-2	3628	6	638	4.61
	-3	4964	6	814	6.11
GEAR	3	227	17	86	1.45
	2	270	20	109	1.76

Table B.14 : Problem C5 (Figure 7.39)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	0	5461	85	365	7.94
	-1	48182	242	2087	25.93
GEAR	0	156	18	32	0.67
	-1	182	18	47	0.80
	-2	226	20	68	1.06
	-3	282	24	101	1.43

Table B.15 : Problem D1 (Figure 7.40)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-1	9060	132	354	10.50
	-2	9249	147	339	7.23
	-3	5367	92	189	4.10
GEAR	-2	170	17	71	0.87
	-3	248	23	116	1.36
	-4	299	26	157	1.79

Table B.16 : Problem D2 (Figure 7.41)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-2	19526	138	142	18.42
	-3	7006	61	401	22.63
	-4	11933	76	794	23.51
GEAR	-3	226	22	93	1.43
	-4	308	27	139	2.10
	-5	396	33	203	2.84
	-6	511	39	280	3.84

Table B.17 : Problem D3 (Figure 7.42)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-2	1643	44	73	1.34
GEAR	-4	54	8	19	0.26
	-5	71	9	24	0.38

Table B.18 : Problem D4

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-2	1528	19	72	1.37
GEAR	-3	162	20	52	0.56
	-4	195	21	68	0.74
	-5	271	25	112	1.09

Table B.19 : Problem D5

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-2	7204	109	392	5.74
	-3	6395	128	357	6.83
GEAR	-3	71	7	26	0.35
	-4	116	11	43	0.59
	-5	150	14	62	0.79
	-6	170	14	80	0.95

Table B.20 : Problem D6 (Figure 7.43)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-4	328	0	43	0.48
	-5	335	0	47	0.51
	-6	368	0	51	0.67
	-7	424	0	60	0.66
GEAR	-3	76	9	15	0.41
	-4	113	12	27	0.63
	-5	161	15	50	0.97
	-6	215	16	86	1.49

Table B.21 : Problem E1 (Figure 7.44)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-2	378	0	55	0.31
	-3	545	0	83	0.46
	-4	795	0	125	0.65
	-5	1188	0	192	0.98
GEAR	0	298	28	124	1.19
	-1	372	28	184	1.58
	-2	440	31	246	2.08
	-3	538	37	339	2.74

Table B.22 : Problem E2 (Figure 7.45)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-2	1971	51	99	1.73
	-3	7097	52	245	5.78
	-4	16599	39	648	13.63
GEAR	-2	165	15	71	0.86
	-3	251	18	121	1.39
	-4	312	21	156	1.81
	-5	405	25	216	2.54

Table B.23 : Problem E3 (Figure 7.46)

Method	Log_{10} Accuracy	FE	JE	Steps	CPU Time
SARK	-4	3779	87	127	5.75
	-5	4965	91	174	6.12
	-6	4355	58	299	5.98
	-7	4701	52	402	6.81
GEAR	-2	290	25	127	1.98
	-3	400	32	202	2.97
	-4	516	38	285	4.10
	-5	669	49	389	5.44

Table B.24 : Problem E4 (Figure 7.47)